650 DATA PROCESSING SYSTEM BULLETIN

BASIC OPERATION CODES, PROGRAM OPTIMIZING, PROGRAM LOADING

This Bulletin obsoletes the following sections of 22-6060-1:

1. Operation Codes
2. Optimum Programming
3. Loading Routines

Operation Codes

The basic operation codes are discussed in these functional groups:

1. Arithmetic Codes
   a. Add
   b. Subtract
   c. Multiply
   d. Divide
2. Store Codes
3. Shift Codes
4. Branching Codes
5. Miscellaneous Codes
6. Table Lookup
7. Absolute Arithmetic Codes

The input-output codes are discussed in 650 M bulletin, Form G24-5001, which covers the three input-output units (533, 537, 407).

Arithmetic Codes

Sign Analysis

In all arithmetic operations, the contents of the storage location specified by the D-address of the instruction is moved first to the distributor. Sign analysis then takes place according to the rules of algebra. Three items enter into sign analysis:

1. Sign of the factor in the accumulator at the beginning of the operation.
2. Type of operation: add, subtract, multiply, divide.
3. Sign of the factor in the distributor (D-address factor).

These items are analyzed to determine the sign of the accumulator at the completion of the operation.

A complete listing of all possible results of sign analysis can be obtained from a basic formula (Figure 1). Figure 2 is a complete listing of these results.

$$[\pm A] \pm \frac{\times}{\div} [\pm B] = [\pm C]$$

SIGN OF ACCUMULATOR AT BEGINNING OF OPERATION	OP CODE	SIGN OF DISTRIBUTOR	SIGN OF ACCUMULATOR AT END OF OPERATION

Figure 1. Basic Sign Analysis Formula

ADD

$(+A)+(+B)=(+C)$
$(+A)+(-B)=(\pm C)^*$
$(-A)+(+B)=(\pm C)^*$
$(-A)+(-B)=(-C)$

SUBTRACT

$(+A)-(-B)=(+C)$
$(+A)-(+B)=(\pm C)^*$
$(-A)-(-B)=(\pm C)^*$
$(-A)-(+B)=(-C)$

MULTIPLY

$(+A)\times(+B)=(+C)$
$(+A)\times(-B)=(-C)$
$(-A)\times(+B)=(-C)$
$(-A)\times(-B)=(+C)$

DIVIDE

$(+A)\div(+B)=(+Q)\ (+R)$
$(+A)\div(-B)=(-Q)\ (+R)$
$(-A)\div(+B)=(-Q)\ (-R)$
$(-A)\div(-B)=(+Q)\ (-R)$

*Sign determined by the larger of the two factors (A, B).

Figure 2. Results of Sign Analysis

15 ALO (Add to Lower). This code causes the contents of the D-address of the instruction to be added to the contents of the lower half of the accumulator. The upper half of the accumulator will be affected by any carries from the high-order position of the lower accumulator.

The 15 ALO code initiates this sequence of steps:

1. Move the contents of the D-address location to the distributor.
2. Make sign analysis.
3. Combine (in the adder) the contents of the distributor with the contents of the lower half of the accumulator, and return the results to the lower accumulator.
4. Combine (in the adder) the contents of the upper half of the accumulator with zeros that are supplied automatically by the machine. The results are returned to the upper accumulator.

Figure 3 shows examples of the results obtained using the 15 ALO code. Each program step is processed in successive blocks. Program step 4 (15-8001-0005) can be executed faster than the other steps, because no movement of data from a storage location to the distributor is necessary.

LOC. OF INSTR.	INSTRUCTION			OPERATION ABBREVIATION
	OP	DATA	INSTR.	
0001	15	0051	0002	ALO
0002	15	0052	0003	ALO
0003	15	0053	0004	ALO
0004	15	8001	0005	ALO
0005	15	8002	0006	ALO
0006	15	8003	XXXX	ALO

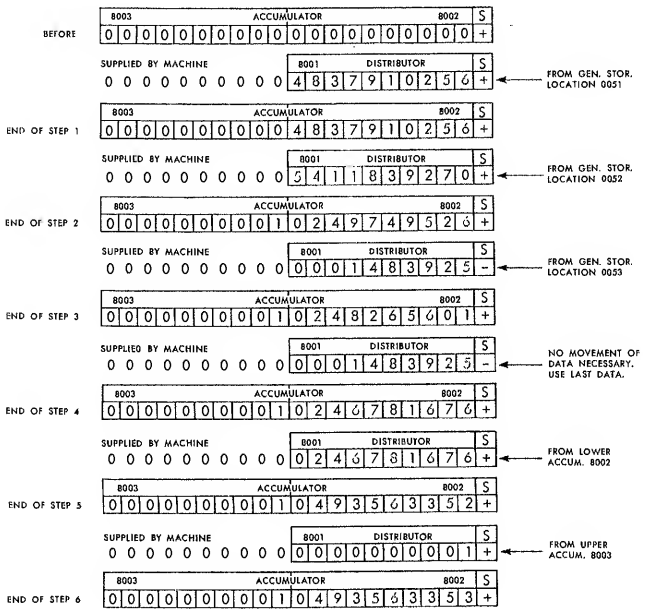


Figure 3. 15 ALO Operation

65 RAL (Reset and Add to Lower). This code is almost identical in function to the 15 ALO code. It causes the contents of the D-address of the instruction to be added into the lower accumulator, after having reset the entire accumulator (20 positions) to zero. The zero accumulator automatically assumes a plus sign.

The 65 RAL code initiates this sequence of steps:

1. Move contents of the D-address location to the distributor.
2. Reset the entire accumulator.
3. Make sign analysis.
4. Combine (in the adder) contents of the distributor with contents (zeros) of the lower accumulator and return the results to the lower accumulator.
5. Combine contents (zeros) of the upper accumulator with the automatically supplied zeros (in the adder) and return the results (zeros) to the upper accumulator.

Figure 4 shows examples of the results obtained using the 65 RAL code. Figure 4D is a method of clearing the upper accumulator to zero and retaining the contents of the lower. Figure 4E is a method of shifting the contents of the upper accumulator ten positions to the right. This method is faster than shifting ten positions using the 30 SRT code.

LOCATION OF INSTRUCTION	INSTRUCTION			OPERATION ABBREV.
	OP	DATA	INSTR.	
0001	65	0051	0002	RAL
0002	65	0053	0003	RAL
0003	65	8001	XXXX	RAL

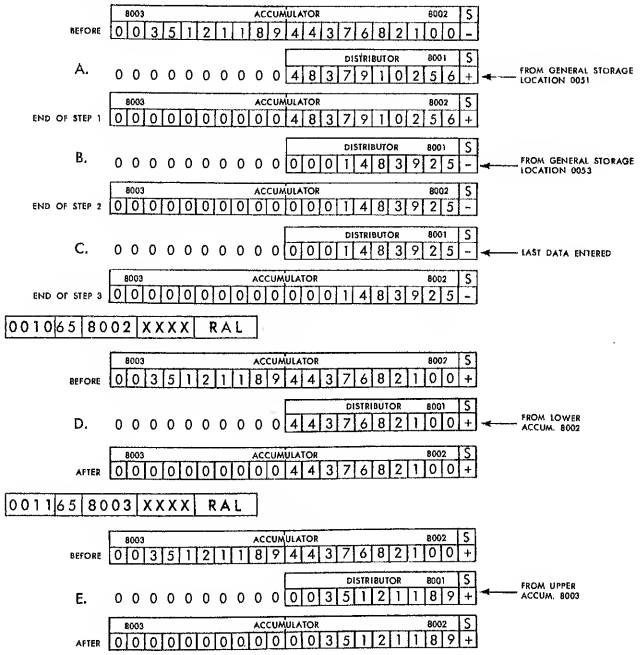


Figure 4. 65 RAL Operation

10 AUP (Add to Upper). This code is similar in function to the 15 ALO code. The only significant difference is that the contents of the D-address location are added to the contents of the upper half of the accumulator. A carry from the high-order position of the upper accumulator will set up an overflow condition that can stop program execution or be interrogated by an operation code.

The 10 AUP code initiates this sequence of steps:

1. Move the contents of the D-address location to the distributor.
2. Make sign analysis.
3. Combine (in the adder) contents of the lower accumulator with zeros supplied automatically by the machine and return the results to the lower accumulator.
4. Combine (in the adder) contents of the upper accumulator with the contents of the distributor and return the results to the upper accumulator.

Figure 5 shows examples of results obtained using the 10 AUP code.

LOCATION OF INSTRUCTION	INSTRUCTION			OPERATION ABBREV.
	OP	DATA	INSTR.	
0052	10	0002	0053	AUP
0053	10	0003	0054	AUP
0054	10	8001	0055	AUP
0055	10	8002	0056	AUP
0056	10	8003	XXXX	AUP

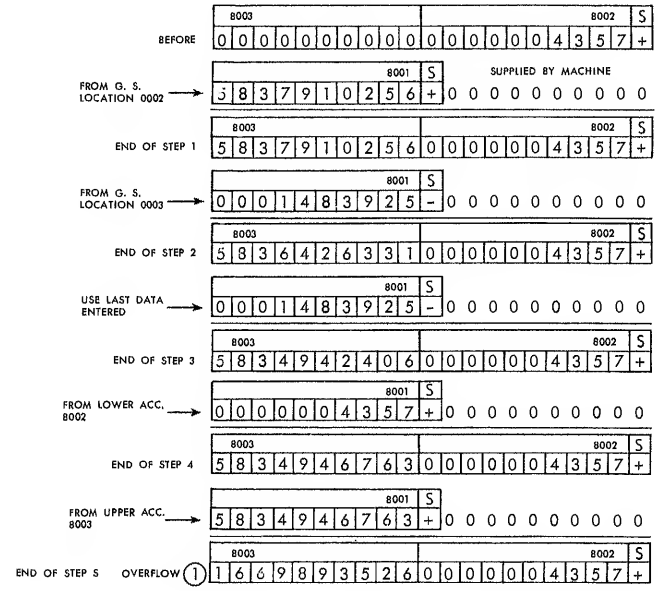


Figure 5. 10 AUP Operation

60 RAU (Reset and Add to Upper). This code is almost identical in function to the 65 RAL code. The only significant difference is that the contents of the D-address location are added into the upper half of the accumulator after resetting the entire accumulator to zero. As in the 65 RAL code, the reset accumulator has a plus sign.

The 60 RAU code initiates this sequence of steps:

1. Move contents of the D-address location to the distributor.
2. Reset the entire accumulator.
3. Make sign analysis.
4. Combine contents of the lower accumulator (zeros) with the automatically supplied zeros and return the results to the lower accumulator.
5. Combine contents of the upper accumulator (zeros) with the contents of the distributor and return the results to the upper accumulator.

Figure 6 shows examples of the results obtained using the 60 RAU code. Figure 6E is a method of clearing the lower accumulator to zero and retaining the contents of the upper. Figure 6D is a method of shifting the contents of the lower accumulator 10 positions to the left. This method is faster than shifting ten positions using the 35 SLT code.

LOCATION OF INSTRUCTION	INSTRUCTION			OPERATION ABBREV.
	OP	DATA	INSTR.	
0001	60	0051	0002	RAU
0002	60	0052	0003	RAU
0003	60	8001	XXXX	RAU

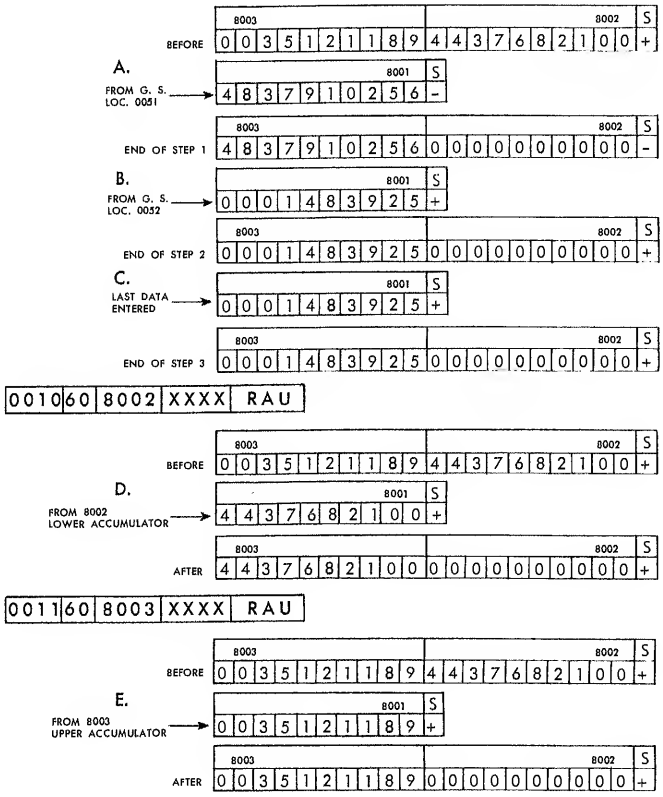


Figure 6. 60 RAU Operation

16 SLO (Subtract from Lower). With the exception of sign analysis, this code functions the same as the 15 ALO code. As in the 15 ALO code, carries from the high-order position of the lower accumulator will affect the upper accumulator. The sequence of steps for this code is the same as the 15 ALO code.

Figure 7 shows examples of results obtained using this code.

LOCATION OF INSTRUCTION	INSTRUCTION			OPERATION ABBREV.
	OP	DATA	INSTRUCTION	
0100	16	0250	0101	SLO
0101	16	0251	0102	SLO
0102	16	0252	XXXX	SLO

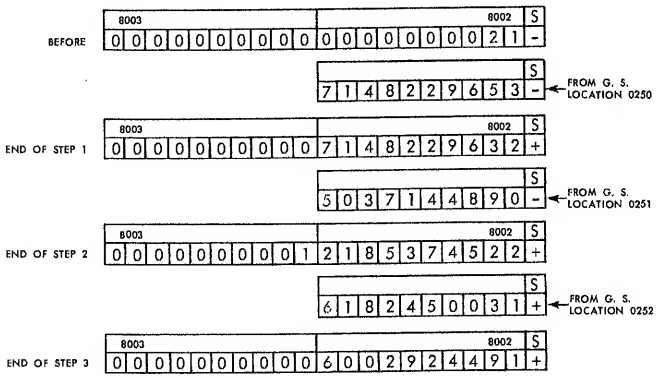


Figure 7. 16 SLO Operation

LOCATION OF INSTRUCTION	INSTRUCTION			OPERATION ABBREV.
	OP	DATA	INSTR.	
1987	66	0097	1124	RSL
1124	66	1127	XXXX	RSL

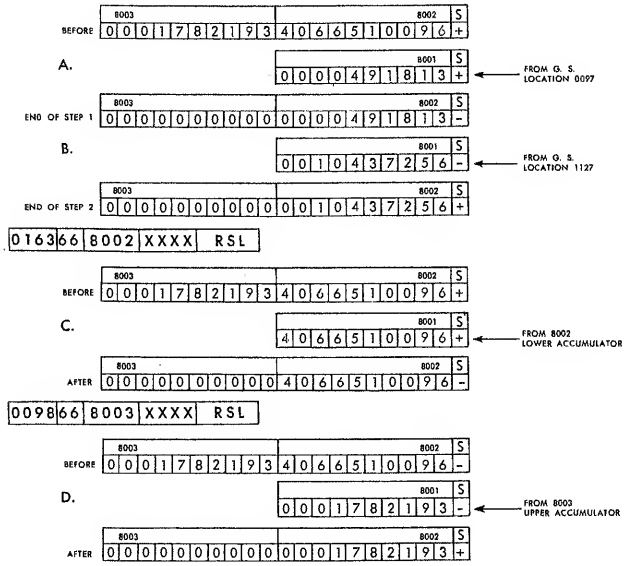


Figure 8. 66 RSL Operation

66 RSL (Reset and Subtract from Lower). With the exception of sign analysis, this code functions the same as the 65 RAL code. Figure 8 shows examples of results obtained using this code. Figure 8C is a method of clearing the upper accumulator and retaining the contents of the lower with a change of sign.

11 SUP (Subtract from Upper). With the exception of sign analysis, this code functions the same as the 10 AUP code. As in the 10 AUP code, carries from the high-order position of the upper accumulator set up an overflow condition. Figure 9 shows examples of results obtained using the 11 SUP code.

LOCATION OF INSTRUCTION	INSTRUCTION			OPERATION ABBREV.
	OP	DATA	INSTR.	
0762	11	0915	1147	SUP
1147	11	0100	0555	SUP
0555	11	0208	XXXX	SUP

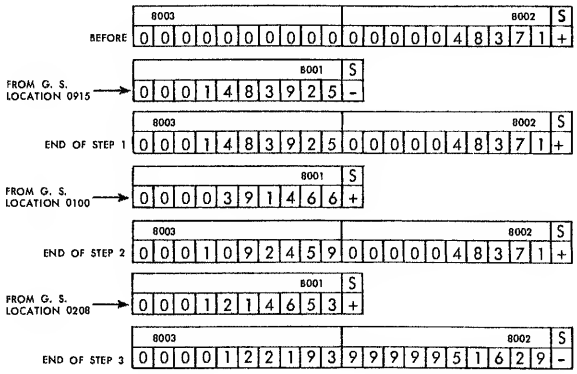


Figure 9. 11 SUP Operation

61 RSU (Reset and Subtract from Upper). With the exception of sign analysis, this code functions the same as the 60 RAU code. Figure 10 shows results obtained using the 61 RSU code. Figure 10D is a method of clearing the lower accumulator and retaining the contents of the upper with a changed sign.

INDEPENDENT ACCUMULATOR OPERATIONS

The upper and lower halves of the accumulator can be used for concurrent independent accumulation. This is made possible because independent operation codes are provided for each half.

To assure that the independent answers are correct, keep in mind that:

1. The sign of both must be the same.
2. The total amount accumulated in each half cannot exceed 10 digits.
3. A reset code (60, 61, 65, 66, 67, 68) resets both halves of the accumulator.

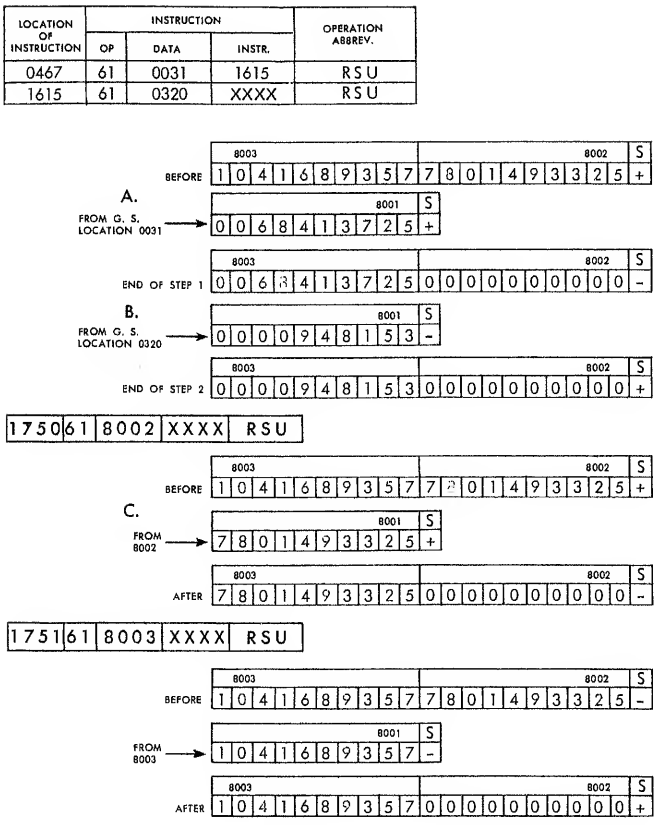


Figure 10. 61 RSU Operation

4. Shifting operations affect both halves of the accumulator.
5. Concurrent independent operations are generally confined to addition, subtraction, and storing.

Figure 9, step 3, is an example of what can occur when an operation on the upper accumulator causes change of sign. If the amounts in the upper and lower halves were independent values, the lower would be incorrect because it is a complement figure.

19 MPY (Multiply). This code causes the 650 to multiply the contents of the location specified by the D-address of the instruction by the contents of the upper accumulator. The maximum size factors are:

1. Multiplier – 10 digits
2. Multiplicand – 10 digits
3. Product – 20 digits

Before the 19 MPY code is used in the program, the multiplier must be placed in the upper accumulator. When a 19 instruction is executed, the multiplicand factor is automatically placed in the distributor (where sign analysis is made) before the actual multiplication is begun. The storage location of the multiplicand is specified by the D-address of the 19 MPY instruction. At the completion of the multiply operation, the product is in the accumulator, the mutiplicand is still in the distributor, and the multiplier is lost. The operations necessary to position factors for decimal-point alignment and to half-adjust results are described under *Shifting*.

The actual multiplication is accomplished by repeated addition of the multiplicand (distributor) under control of the multiplier (upper accumulator). The general sequence of steps of the actual multiplication (Figure 11) is:

1. Shift the contents of the entire accumulator one position to the left. This places the high order position of the multiplier in a special storage position where it is analyzed.
2. Add the contents of the distributor (multiplicand) into the lower accumulator twice.
3. Shift the contents of the entire accumulator one position to the left. This places the ninth position of the multiplier (zero) in the special storage position for analysis.
4. Because of the zero, no addition of the distributor is necessary. Therefore, another left shift of one position is taken. This continues through the five remaining zeros.
5. When the third position of the multiplier (5) is placed in the analysis position, the distributor is added into the lower accumulator five times.
6. Left-shift the accumulator one position and analyze the 4. Shifting, analyzing, and adding continue until all ten positions of the multiplier have been used.

The time taken to complete a multiply operation depends upon the sum of all digits in the multiplier. In Figure 11, a total of 19 additions is required to complete the operation. This is true for any arrangement of the digits (2, 5, 4, 8). If the factors in Figure 11 (distributor – upper accumulator) are reversed, the operation is completed sooner because only 10 additions are required.

The sequence of steps initiated by a 19 MPY code is:

1. Move the contents of the D-address location to the distributor.
2. Sign analysis.
3. Actual multiplication.

```
  1 1 1 1 1 1 1 1 1 1
X 2 0 0 0 0 0 0 5 4 8
---------------------
2 2 2 2 2 2 2 8 3 0 8 8 8 8 8 8 2 8
```

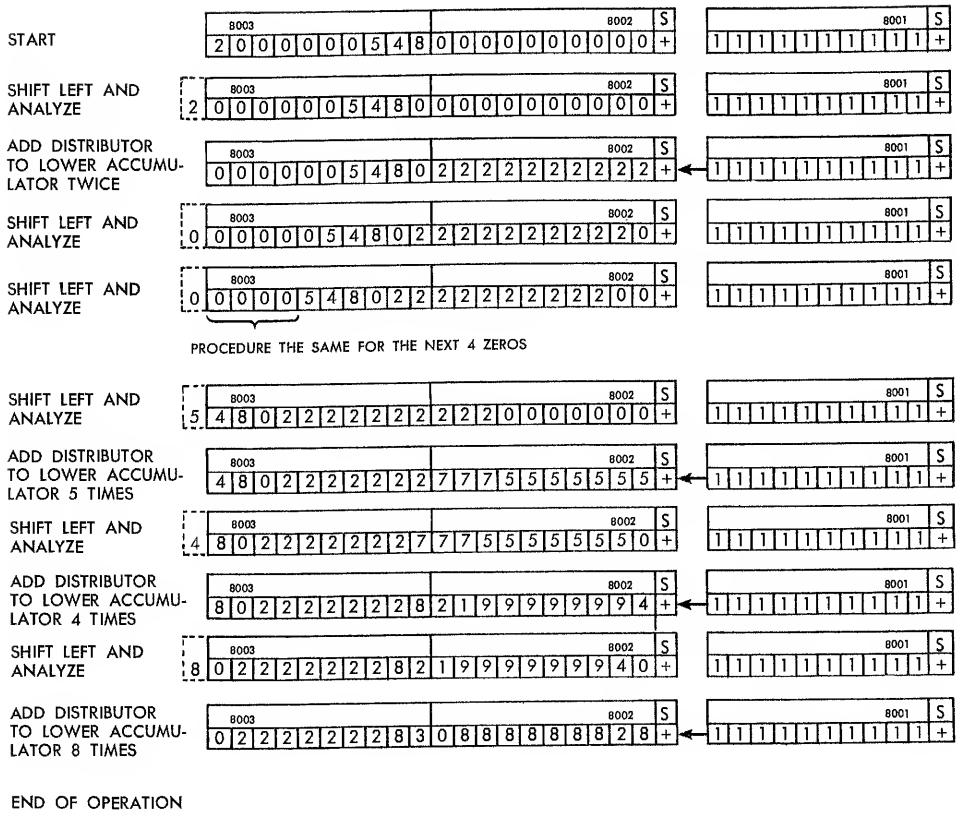


Figure 11. General Multiplication Operation

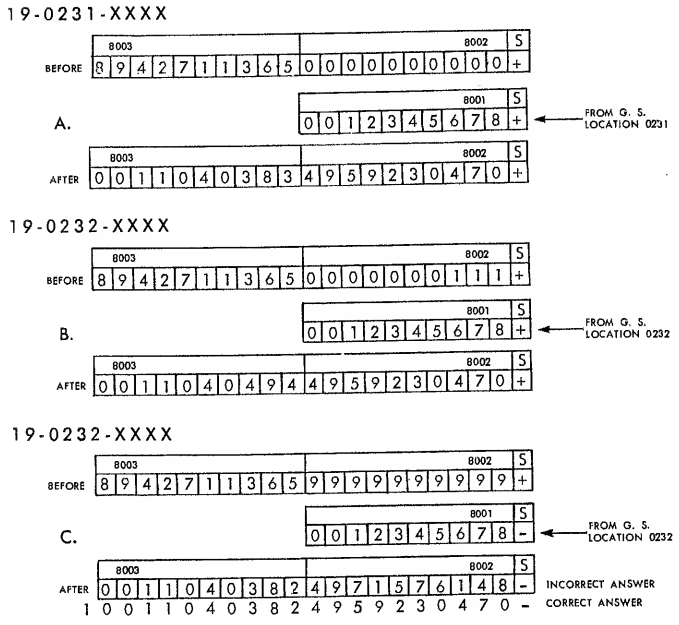


Figure 12. 19 MPY Operation

If the lower half of the accumulator contains some number at the start of the multiply operation, the absolute value of that number will be added to the absolute value of that portion of the product developed in the upper half of the accumulator (Figure 12B). While multiplying, the regular rules of carry apply. Therefore, a sufficiently large number in the lower accumulator could increase the value of the multiplier, resulting in an erroneous answer (Figure 12C).

14 DIV (Divide). This code causes the 650 to divide the contents of the entire accumulator by the contents of the location specified by the D-address of the instruction. The remainder is retained in the upper half of the accumulator at the completion of the operation. The quotient is developed in the lower half of the accumulator. The dividend factor is lost.

64 DVR (Divide and Reset Upper). This code is identical to the 14 DIV code except that the remainder is *not* retained. At the completion of the 64 DVR operation, the upper accumulator is zero, and the lower accumulator contains the quotient.

The maximum size factors are:

1. Dividend – 20 digits (accumulator) *
2. Divisor – 10 digits (distributor) *
3. Quotient – 10 digits (lower accumulator)
4. Remainder – 10 digits (upper accumulator)

*The absolute value of the divisor must be greater than the absolute value of that part of the dividend that is in the upper half of the accumulator. If this rule is not observed, a quotient overflow occurs and program execution halts. Also, a quotient overflow occurs when a division by zero is attempted. When program execution halts, the program register can contain either the divide (14, 64) instruction or the next instruction of the program. If the overflow occurs *before* the next instruction is entered into the program register, the divide instruction will still be in the program register when program execution stops. If the overflow occurs *after* the next instruction is entered into the program register, the next instruction is in the program register when program execution stops.

Before either divide code (14, 64) is used in the program, the dividend must be placed in the accumulator. The dividend can be shifted for decimal point alignment prior to the actual division, provided the preceding rule (*) is observed. When the divide instruction is analyzed, the divisor is automatically placed in the distributor prior to the beginning of the actual division. The storage location of the divisor is specified by the D-address of the divide instruction.

The sequence of steps, initiated by a divide code, is:

1. Move contents of the D-address location to the distributor
2. Sign analysis
3. Actual division.

At the completion of the divide operation, the divisor is still in the distributor, the quotient is in the lower half of the accumulator, the remainder (if retained) is in the upper half of the accumulator, and the dividend is lost. Figures 13 and 14 are examples of results obtained using the 14 DIV and 64 DVR codes.

Because it is possible for the quotient and remainder to have the same or different signs, each half of the accumulator will have its own sign at the completion of a divide operation. This is the only condition under which the upper accumulator can have a sign different from the lower accumulator. The dual sign condition remains until another arithmetic reset, multiply, or divide operation is performed. At this point, the remainder sign is lost and the entire accumulator reverts to the sign of the quotient.

For this reason, the remainder should be stored on the drum before the next arithmetic operation. When the remainder is stored under these conditions, it is stored with the correct sign. If a multiply operation

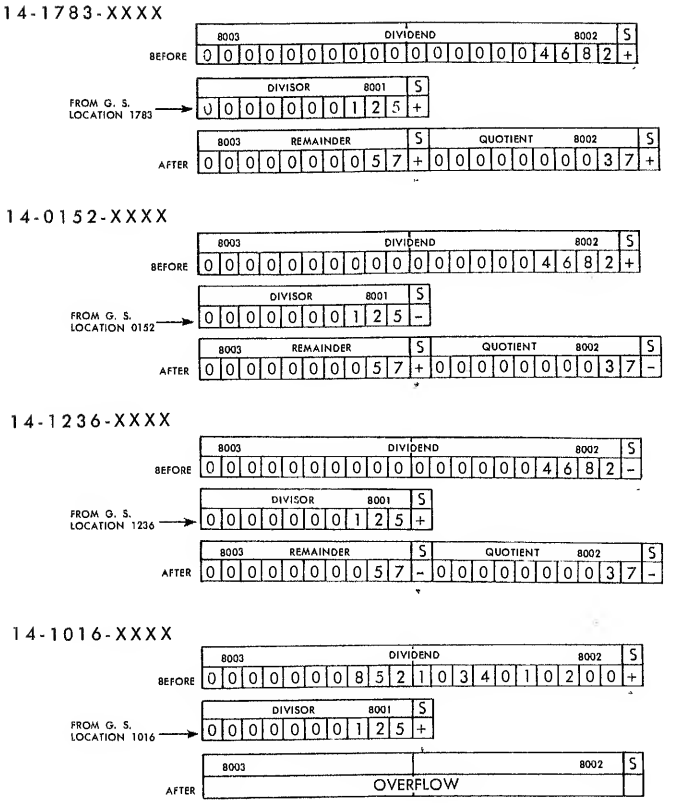


Figure 13. 14 DIV Operation

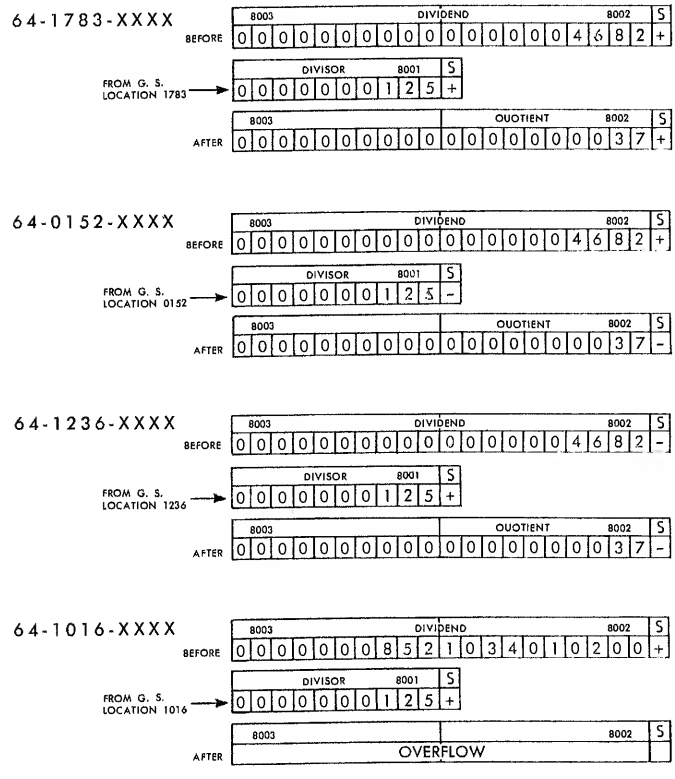


Figure 14. 64 DVR Operation

is executed any time after a 14 DIV operation, but before a reset operation, the multiplier sign is placed in the remainder sign position. If a store-upper-accumulator code is executed after the multiply, that part of the product in the upper accumulator is stored with the sign of the multiplier, rather than the sign of the product. If the branch-minus code (46 BMI) is used after a divide operation, the sign of the quotient is tested and not the sign of the remainder.

Division in the 650 is accomplished in a manner almost opposite to that of multiplication. The divisor factor (distributor) is subtracted from that part of the dividend factor which is in the upper accumulator. The count of the number of subtractions is kept in the lower accumulator and becomes the quotient. As in multiplying, shifting the contents of the accumulator to the left plays an important part in the operation.

Figure 15 show the general sequence of steps:

1. The contents of the distributor are successively subtracted from the contents of the upper accumulator until an overdraw occurs. After the overdraw occurs, the distributor contents are added back into the upper accumulator. The number of subtractions before the overdraw is entered into the units position of the lower accumulator as the first digit of the quotient.
2. Shift the entire accumulator one position to the left.
3. Repeat step 1.
4. Repeat step 2. Shifting and subtraction continue until ten shift cycles have been taken.

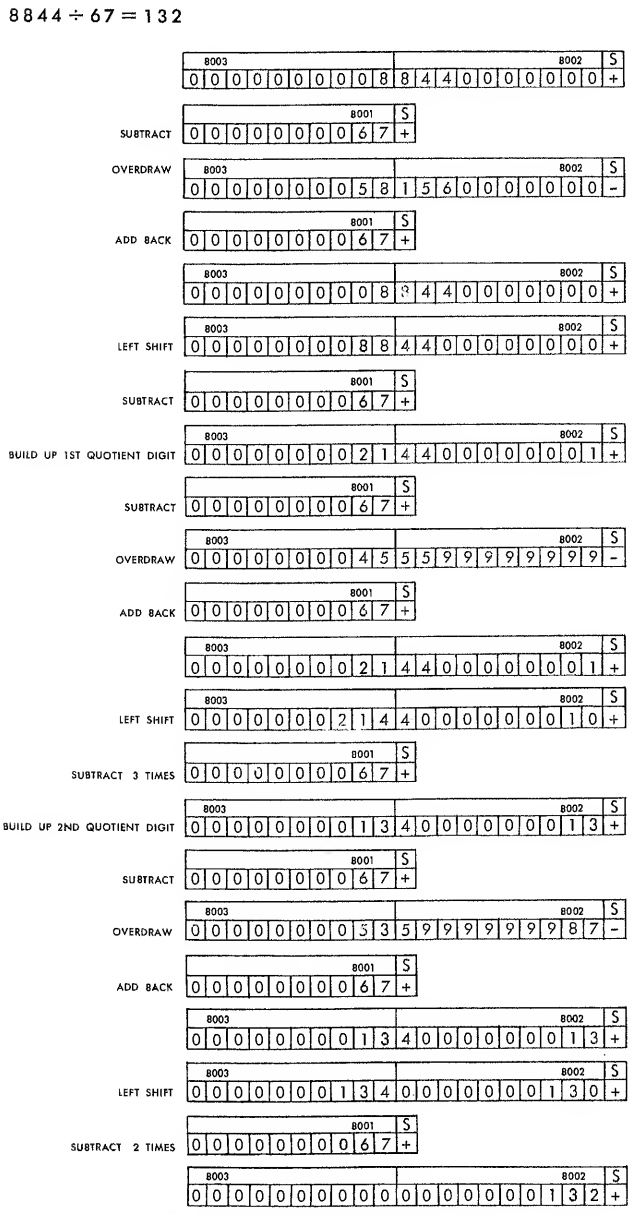


Figure 15. General Divide Operation

Store Codes

These codes move information from the process section (accumulator-distributor) to some location in general storage. Information moving from the accumulator to general storage passes through the distributor. This replaces the former contents of the distributor with the information being stored. In general, store codes operate this way:

1. The Op code determines what unit (upper accumulator, lower accumulator, distributor) is to be stored.
2. The D-address designates the location in general storage where the information is to go. Only general storage location addresses (0000-1999) are valid when using store codes (the addition of Immediate Access Storage and Indexing Registers increases the range of valid addresses).

Moving information from one storage location to another does not destroy or modify the contents of the sending location. Only the receiving location has its information changed. Any storage unit can be read from, as often as necessary, without affecting its contents.

20 STL (Store Lower Accumulator). This code causes the contents of the lower half of the accumulator, with its sign, to be placed in the general storage location specified by the D-address of the instruction. The 20 STL code initiates this sequence of steps:

1. Move contents of lower accumulator to distributor.
2. Move the new contents of the distributor to the general storage location specified by the D-address of the 20 STL instruction.

Figure 16 is an example of the use of the 20 STL code.

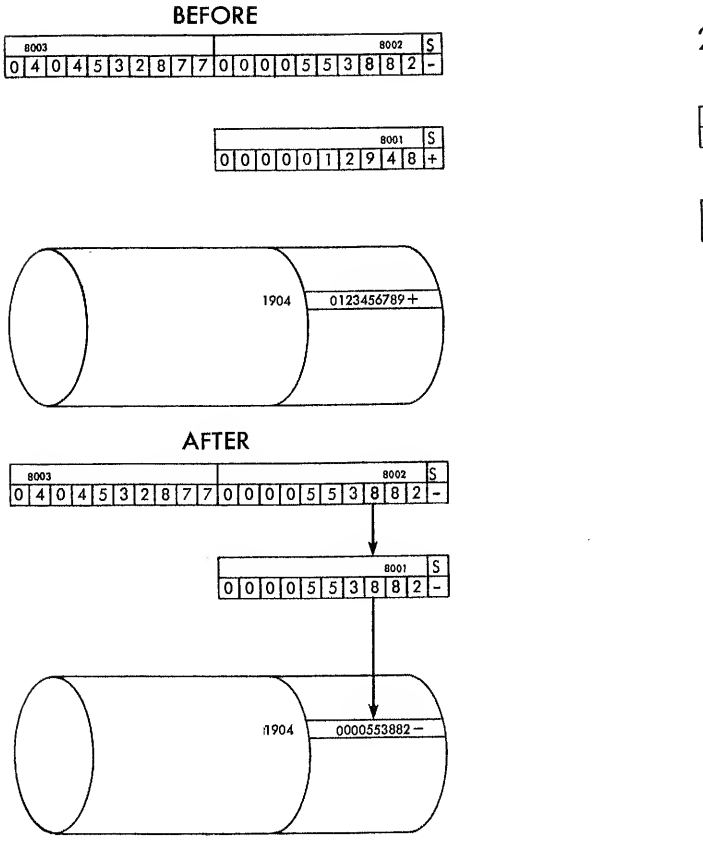


Figure 16. 20 STL Operation

21 STU (Store Upper Accumulator). This code causes the contents of the upper half of the accumulator to be placed in the general storage location specified by the D-address of the instruction. Normally, the sign stored is that of the entire accumulator. However, if the 21 STU is performed following a 14 DIV operation, and before another division, multiplication, or reset operation takes place, the sign stored is that of the remainder and *not* the sign of the quotient. Figure 17 shows examples of results obtained using the 21 STU code.

22 SDA (Store Data Address of Lower Accumulator). This code causes positions 8-5 of the lower accumulator to be moved to the distributor, replacing the former contents of positions 8-5 of distributor. The contents of the distributor are then moved to the general storage location specified by the D-address of the instruction. The sign stored in this operation is the sign of the distributor and *not* the sign of the accumulator. In general, this operation code has its major use in the process of instruction-modification.

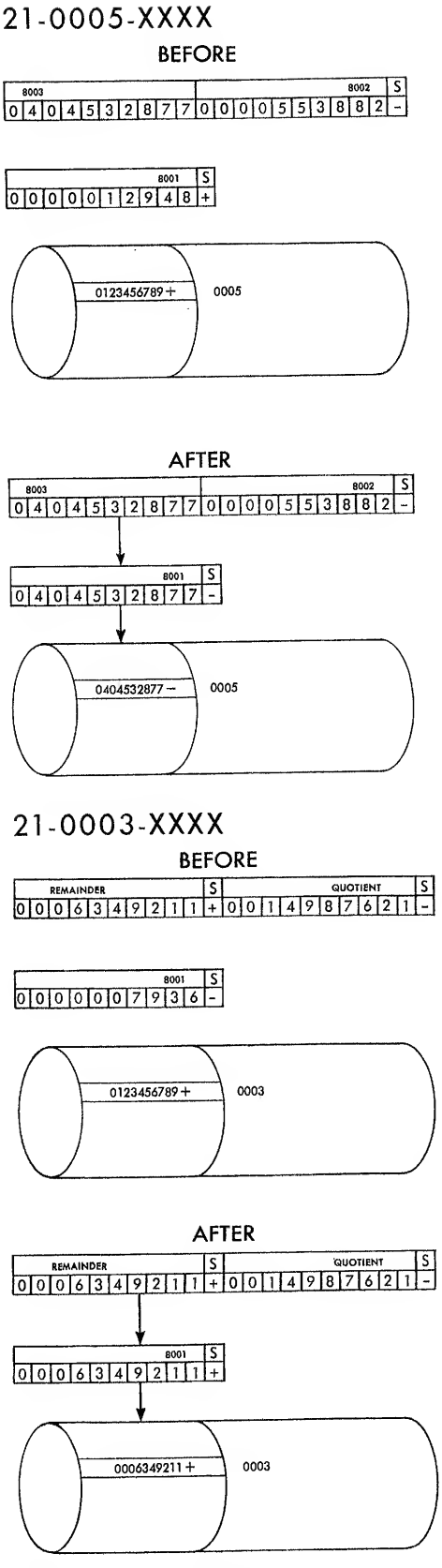


Figure 17. 21 STU Operation

The sequence of steps initiated by a 22 SDA instruction is:

1. Move contents of positions 8-5 of lower accumulator to positions 8-5 of distributor. (The only information changed in distributor is in positions 8-5).
2. Move contents of the entire distributor, with its sign, to the general storage location specified by the D-address of the instruction.

Figure 18 shows results obtained using the 22 SDA code.

22-1955-XXXX

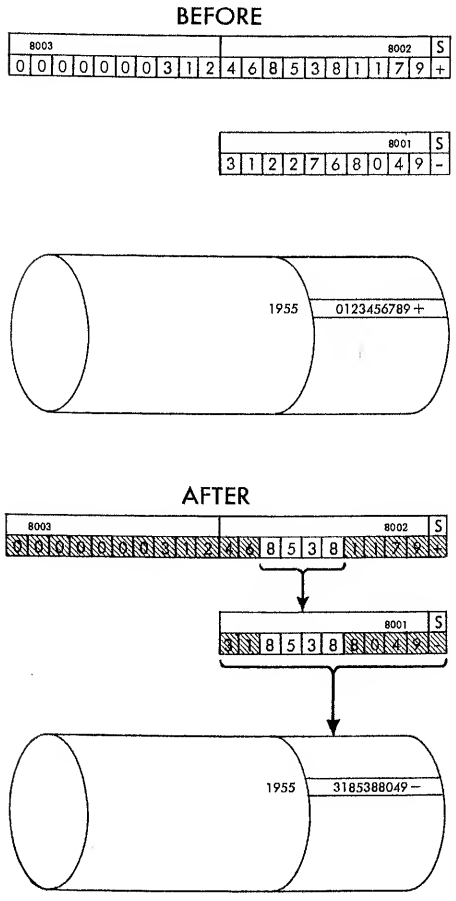


Figure 18. 22 SDA Operation

As in the 22 SDA code, this code is used in instruction-modification.

The sequence of steps, initiated by a 23 SIA code, is:

1. Move contents of positions 4-1 of lower accumulator to positions 4-1 of distributor.
2. Move contents of entire distributor, with its sign, to the general storage location specified by the D-address of the instruction.

Figure 19 shows an example of the results obtained when using the 23 SIA code.

23-1955-XXXX

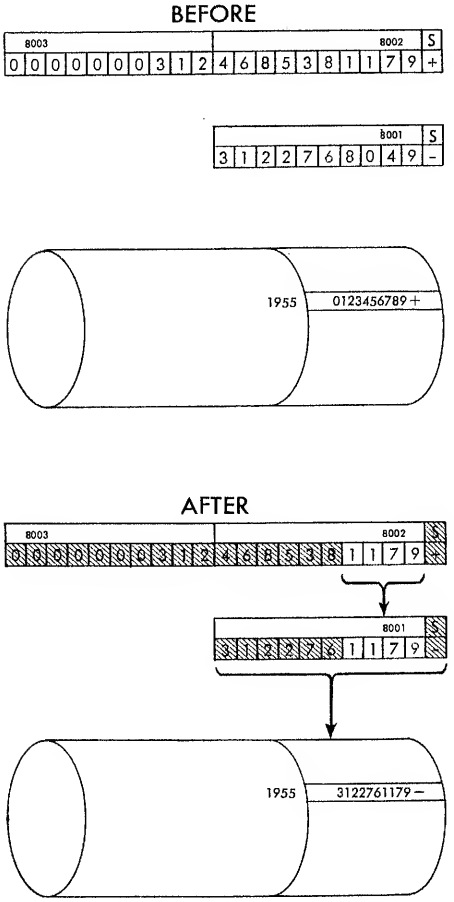


Figure 19. 23 SIA Operation

23 SIA (Store Instruction Address of Lower Accumulator). This code causes positions 4-1 of the lower accumulator to be moved to the distributor, replacing the contents of positions 4-1 of the distributor. The entire contents of the distributor, with its sign, are then placed in the general storage location specified by the D-address of the 23 SIA instruction. As in the 22 SDA code, the sign stored with this operation is the sign of the distributor, and not of the accumulator.

24 STD (Store Distributor). This code causes the contents of the distributor, with its sign, to be placed in the general storage location specified by the D-address of the instruction.

Shift Codes

All shifting operations are accomplished within the accumulator. The contents of both halves of the accumulator are shifted right or left as indicated by the

Op code. All positions made vacant by the shift are filled with zeros. The sign of the accumulator is not affected by the shift operation. The distributor is not affected by the shift operation.

The shifting of all significant digits of a negative number out of the accumulator will result in a minus zero. Shifting after a 14 DIV operation (before a reset, multiply, or divide) does not change the signs associated with the upper and lower halves of the accumulator, and may also result in a minus zero.

In any shift operation two things must be considered:

1. Which way (right or left) is the accumulator to be shifted?
2. How many positions is it to shift?

The operation code determines the direction of the shift, and the units position of the D-address determines the number of positions shifted. In all cases the entire D-address of a shift instruction must be valid.

30 SRT (Shift Right). This operation code causes the contents of the entire accumulator to be shifted to the right. The number of places shifted is specified by the units digit of the D-address. A maximum shift of 9 positions is possible. A D-address with a units digit of zero will result in no shift. All numbers shifted off the right end of the accumulator are lost. The left-hand positions are filled with zeros as the shift takes place.

Figure 20 gives examples of results obtained when using the 30 SRT code.

30-1832-XXXX

	8003										8002										S
BEFORE	9	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8	9	9	-
AFTER	0	0	9	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8	-

30-1830-XXXX

	8003										8002										S
BEFORE	9	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8	9	9	-
AFTER	9	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8	9	9	-

Figure 20. 30 SRT Operation

31 SRD (Shift Right and Round). This operation code causes the contents of the entire accumulator to be shifted to the right. The number of positions shifted is specified by the units digit of the D-address. At the completion of the shift, a 5 is added to the last digit shifted out of the accumulator (minus 5 if the accumulator sign is negative) to half adjust the amount in the accumulator. A D-address with a units digit of 1-9 causes a shift of 1-9, respectively, with rounding. A D-address with units digit of 0 will result in a right shift of 10 with rounding.

Figure 21 shows results obtained when using the 31 SRD code.

31-1832-XXXX

	8003										8002										S	
BEFORE	9	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8	9	7	+	9
DURING	0	0	9	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8	+	+5
AFTER	0	0	9	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	9	+	

31-1830-XXXX

	8003										8002										S	
BEFORE	9	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8	9	9	+	1
DURING	0	0	0	0	0	0	0	0	0	0	9	9	8	7	6	5	4	3	2	1	+	+5
AFTER	0	0	0	0	0	0	0	0	0	0	9	9	8	7	6	5	4	3	2	1	+	

Figure 21. 31 SRD Operation

35 SLT (Shift Left). This code causes the contents of the entire accumulator to be shifted to the left. The number of places shifted is specified by the units digit of the D-address. A maximum shift of 9 positions is possible. A D-address with a units digit of 0 results in no shift. All numbers shifted out of the left end of the accumulator are lost. Zeros are inserted at the right end of the accumulator as the shifting takes place. Digits shifted out of the left end of the accumulator do not cause an overflow condition.

Figure 22 shows results obtained when using the 35 SLT code.

35-0436-XXXX

	8003										8002										S
BEFORE	9	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8	9	9	+
AFTER	4	3	2	1	1	2	3	4	5	6	7	8	9	9	0	0	0	0	0	0	+

35-0430-XXXX

	8003										8002										S
BEFORE	9	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8	9	9	+
AFTER	9	9	8	7	6	5	4	3	2	1	1	2	3	4	5	6	7	8	9	9	+

Figure 22. 35 SLT Operation

36 SCT (Shift Left and Count). This operation code has multiple functions as follows:

1. The contents of the entire accumulator are shifted to the left. The maximum number of left shifts that can be taken is determined by the units digit of the D-address and the position (counting from left to right) of the first significant digit (1-9) in the upper accumulator. Up to 10 shifts are possible. The number of shifts taken can be less than that specified by the units position of the D-address but never more. A units

digit of zero indicates a maximum of ten shifts. A units digit of nine indicates a maximum of nine shifts, etc.

2. A count of the number of positions that the accumulator shifted to the left is placed in the 2 low order positions of the accumulator at the completion of the shift operation. The value of the count number is determined by the number of shifts taken and the units digit of the data address of the shift instruction. If the units digit of the D-address is zero, the two low-order positions of the accumulator will contain a number that indicates, in true form, the exact number of shifts taken (Figure 23). If the units digit of the D-address is some number other than zero (1-9) the count number placed in the 2 low-order positions of the accumulator is determined this way:

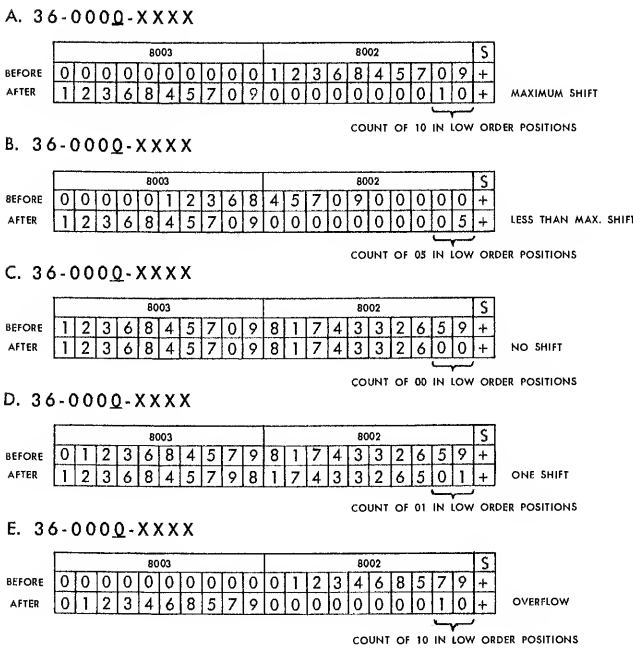


Figure 23. 36 SCT Operation—Maximum 10 Shifts

   a. The tens complement of the units position is used for the base of the count.
   b. The number of shifts taken is added to the tens complement (Figure 24 B, D).

   If a shift and count operation is called for and no shift takes place, the two low-order digits of the accumulator are replaced by zeros regardless of the value in the units position of the D-address (Figure 23C; 24C). If a shift and count operation is called for and only one shift is executed, the units digit of the original accumulator contents is replaced by zero (Figure 23C; 24D).

The SCT operation can be stopped in two ways:

1. A digit, other than zero, is sensed in the high-order position of the accumulator. This stop takes place whenever the number of shifts taken is EQUAL TO or LESS THAN the number of shifts called for by the units digit of the D-address (Figure 23A, B, C, D; 24A, B, C, D).
2. No significant digit has been sensed in the high-order position of the accumulator by the time the accumulator has been shifted the number of positions called for by the D-address. This type of stop results in an overflow condition. This overflow condition can be interrogated at the discretion of the programmer by the Branch Overflow code. If an overflow condition occurs during any 36 SCT operation, the number 10 will be placed in the two low-order positions of the accumulator (Figures 23E; 24E).

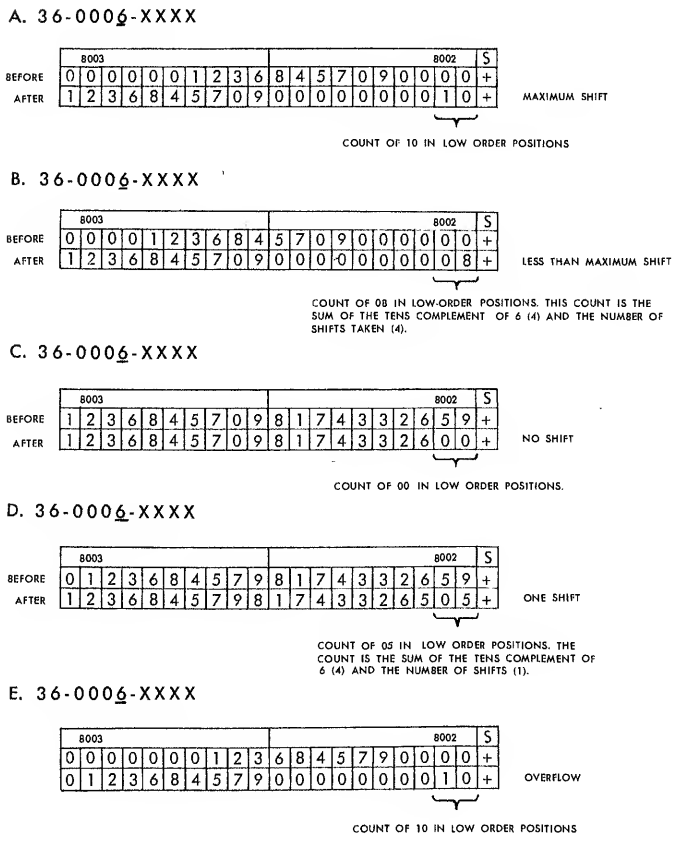


Figure 24. 36 SCT Operation—Maximum Less Than 10 Shifts

Branching Codes

Branching codes provide the 650 with a means of making logical decisions. Branching codes make it possible for the machine to determine which one of two areas of the program is to be used to continue the processing. Branching codes, in effect, ask a question

that can be answered YES or NO. The answer determines where the next instruction of the program is located.

An example of the use of a branching code in a program is a payroll application. It must be determined if the employee has reached the maximum FICA tax that he must pay this year. The answer determines which way his earnings are processed. Figure 25 is a partial block diagram of this part of a payroll job.

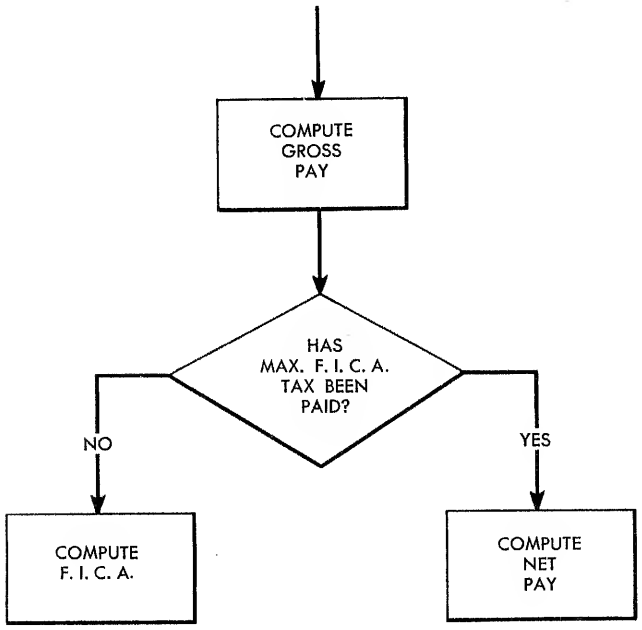


Figure 25. Branching Point

A branch code consists of three parts: the two digit OP CODE (determines the area to be interrogated) and two possible choices for the location of the next instruction. When the answer to the question is YES, the next instruction is specified by the D-address. If the answer to the question is NO, the next instruction is specified by the I-address.

44 NZU (Branch on Non-Zero in Upper Accumulator). This code causes the contents of the upper accumulator to be examined for zero. In effect, the question asked is "Is there any significant digit (1-9) in the upper accumulator?" If there is, the location of the next instruction to be executed is specified by the D-address. If the contents of upper accumulator are zero, the location of the next instruction to be executed is specified by the I-address. The sign of the accumulator is ignored. Figure 26 gives examples of the results obtained using this code.

A

LOCATION OF INSTR.	INSTRUCTION OP	DATA	INSTR.	OPERATION ABBREV.	8003 UPPER ACCUMULATOR	8002 LOWER ACCUMULATOR	SIGN
0152	44	0155	0156	* NZU	0 0 1 2 3 4 5 7 8 6	8 0 7 0 6 4 3 2 9 5	+
0155	21	1982	0156	STU	0 0 1 2 3 4 5 7 8 6	8 0 7 0 6 4 3 2 9 5	+
0156	20	1983	XXXX	STL	0 0 1 2 3 4 5 7 8 6	8 0 7 0 6 4 3 2 9 5	+

*Next instruction located at 0155 (Branch)

B

0152	44	0155	0156	* NZU	0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0	+
0155	21	1982	0156	STU	0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0	+
0156	20	1983	XXXX	STL	0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0	+

*Next instruction located at 0155 (Branch)

C

0152	44	0155	0156	* NZU	0 0 1 2 3 4 5 6 7 8	0 0 0 0 0 0 0 0 0 0	-
0155	21	1982	0156	STU	0 0 1 2 3 4 5 6 7 8	0 0 0 0 0 0 0 0 0 0	-
0156	20	1983	XXXX	STL	0 0 1 2 3 4 5 6 7 8	0 0 0 0 0 0 0 0 0 0	-

*Next instruction located at 0155 (Branch)

D

0152	44	0155	0156	* NZU	0 0 0 0 0 0 0 0 0 0	8 0 7 0 6 4 3 2 9 5	+
0155	21	1982	0156	STU	0 0 0 0 0 0 0 0 0 0	8 0 7 0 6 4 3 2 9 5	+
0156	20	1983	XXXX	STL	0 0 0 0 0 0 0 0 0 0	8 0 7 0 6 4 3 2 9 5	+

*Next instruction located at 0156 (No Branch)

E

0152	44	0155	0156	* NZU	0 0 0 0 0 0 0 0 0 0	8 0 7 0 6 4 3 2 9 5	-
0155	21	1982	0156	STU	0 0 0 0 0 0 0 0 0 0	8 0 7 0 6 4 3 2 9 5	-
0156	20	1983	XXXX	STL	0 0 0 0 0 0 0 0 0 0	8 0 7 0 6 4 3 2 9 5	-

*Next instruction located at 0156 (No Branch)

Figure 26. 44 NZU Operation

A.

LOCATION OF INSTRUCTION	INSTRUCTION			OPERATION ABBREV.	8003 UPPER ACCUMULATOR	8002 LOWER ACCUMULATOR	SIGN
	OP	DATA	INSTRUCTION				
1800	45	1804	1812	* NZE	0 0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0 0	+
1804	21	0008	1811	STU	0 0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0 0	+
1811	20	0009	1812	STL	0 0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0 0	+

*Next instruction located at 1804 (Branch)

B.

1800	45	1804	1812	* NZE	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1 2 3	+
1804	21	0008	1811	STU	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1 2 3	+
1811	20	0009	1812	STL	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1 2 3	+

*Next instruction located at 1804 (Branch)

C.

1800	45	1804	1812	* NZE	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	-
1804	21	0008	1811	STU	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	-
1811	20	0009	1812	STL	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	-

*Next instruction located at 1812 (No Branch)

Figure 27. 45 NZE Operation

A.

LOCATION OF INSTRUCTION	INSTRUCTION			OPERATION ABBREV.	8003 UPPER ACCUMULATOR	8002 LOWER ACCUMULATOR	SIGN
	OP	DATA	INSTRUCTION				
0150	46	0153	0154	* BMI	0 0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0 0	-
0153	21	0027	0180	STU	0 0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0 0	-
0154	21	0077	0180	STU	0 0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0 0	-

*Next instruction located at 0153 (Branch)

B.

0150	46	0153	0154	* BMI	0 0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0 0	+
0153	21	0027	0180	STU	0 0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0 0	+
0154	21	0077	0180	STU	0 0 0 0 0 0 0 0 1 2 3	0 0 0 0 0 0 0 0 0 0 0	+

*Next instruction located at 0154 (No Branch)

C.

0150	46	0153	0154	* BMI	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1 2 3	-
0153	20	0027	0180	STL	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1 2 3	-
0154	20	0077	0180	STL	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1 2 3	-

*Next instruction located at 0153 (Branch)

D.

0150	46	0153	0154	* BMI	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1 2 3	+
0153	20	0027	0180	STL	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1 2 3	+
0154	20	0077	0180	STL	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1 2 3	+

*Next instruction located at 0154 (No Branch)

E.

0150	46	0153	0154	* BMI	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	-
0153	20	0027	0180	STL	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	-
0154	20	0077	0180	STL	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	-

*Next instruction located at 0153 (Branch)

F.

0150	46	0153	0154	* BMI	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	+
0153	20	0027	0180	STL	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	+
0154	20	0077	0180	STL	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	+

*Next instruction located at 0154 (No Branch)

Figure 28. 46 BMI Operation

A.

LOCATION OF INSTRUCTION	INSTRUCTION			OPERATION ABBREV.	8002 UPPER ACCUMULATOR	8002 LOWER ACCUMULATOR	SIGN	8001 DISTRIBUTOR	SIGN
	OP	DATA	INSTR.						
0100	60	0103	0157	RAU	8,4,3,1,5,6,0,7,9,2	0,0,0,0,0,0,0,0,0,0	+	8,4,3,1,5,6,0,7,9,2	+
0157	10	0110	0115	AUP	2,0,4,9,9,8,9,8,6,7	0,0,0,0,0,0,0,0,0,0	+	3,6,1,8,4,2,9,0,7,5	+
0115	47	0118	0120	* BOV	2,0,4,9,9,8,9,8,6,7	0,0,0,0,0,0,0,0,0,0	+	3,6,1,8,4,2,9,0,7,5	+

*Next instruction located at 0118 (Branch)

B.

0100	65	0103	0157	RAL	0,0,0,0,0,0,0,0,0,0	8,4,3,1,5,6,0,7,9,2	+	8,4,3,1,5,6,0,7,9,2	+
0157	15	0110	0115	ALO	0,0,0,0,0,0,0,0,0,1	2,0,4,9,9,8,9,8,6,7	+	3,6,1,8,4,2,9,0,7,5	+
0115	47	0118	0120	* BOV	0,0,0,0,0,0,0,0,0,1	2,0,4,9,9,8,9,8,6,7	+	3,6,1,8,4,2,9,0,7,5	+

*Next instruction located at 0120 (No Branch)

Figure 29. 47 BOV Operation

45 NZE (Branch on Non-Zero Accumulator). This code causes the contents of the entire accumulator to be examined for zero. In effect, the question asked is "Is there any significant digit (1-9) in the entire accumulator?" If there is, the location of the next instruction is specified by the D-address. If the contents of the entire accumulator are zero, the location of the next instruction is specified by the I-address. The sign of the accumulator is ignored. Figure 27 is an example of the use of the 45 NZE code.

46 BMI (Branch on Minus Accumulator). This code causes the sign of the accumulator to be examined for minus. In effect, the question asked is "Is the sign of the accumulator minus?" If it is, the next instruction to be executed is specified by the D-address. If it is plus, the next instruction to be executed is specified by the I-address. The contents of the accumulator are ignored.

If this branch code is used after a divide (14-DIV) operation, the sign of the quotient is tested, and *not* the sign of the remainder.

Keep in mind, when using the 46 BMI code, that it is possible for the accumulator to be both zero and minus at the same time.

Figure 28 shows results obtained using the 46 BMI code.

47 BOV (Branch on Overflow). This code causes the machine to examine the accumulator to see if an overflow condition has occurred. A verifiable overflow condition can be set up by exceeding the capacity of the accumulator on arithmetic operations (plus or minus), or under those conditions described in the 36 SCT code. A non-verifiable overflow condition (which will always stop the machine) can result from division if the machine tries to develop a quotient of more than 10 digits.

The question asked by the 47 BOV is "Has an overflow occurred?" If it has, the next instruction to be executed is specified by the D-address. If no overflow occurred, the location of the next instruction is specified by the I-address.

A further condition is necessary for this instruction to be effective. On the control console is a switch labeled OVERFLOW, with two positions: SENSE and STOP. With this switch set to STOP, an overflow will cause an immediate stop of program execution. With the switch set to SENSE, the 47 BOV code is effective. Also, when any overflow occurs, the OVERFLOW light on the control console is lighted. Execution of the 47 BOV code resets the overflow condition and turns off the OVERFLOW light. Another overflow must occur before the circuit is reactivated. Figure 29 illustrates the results of the 47 BOV code.

90-99 BD 0-9 (Branch on Digit Eight in a Distributor Position). This code causes the machine to examine a particular position of the distributor for the presence of an eight or nine. Codes 91-99 test positions 1-9 of the distributor; code 90 tests position 10 (high order). If an eight is present, the location of the next instruction is specified by the D-address. If a nine is present, the location of the next instruction is specified by the I-address. The presence of any digit other than 8 or 9 causes program execution to stop. Figure 30 illustrates the results obtained using the Branch Distributor codes.

LOCATION OF INSTR.	INSTRUCTION			OPERATION ABBREV.	8001 DISTRIBUTOR	SIGN
	OP	DATA	INSTR.			
0500	69	0010	0513	LDD	0,0,0,0,0,0,0,7,9,8	+
0513	91	0516	0518	* BD1	0,0,0,0,0,0,0,7,9,8	+
0518	92	0521	0523	** BD2	0,0,0,0,0,0,0,7,9,8	+
0523	93	0526	0528	** BD3	0,0,0,0,0,0,0,7,9,8	+
0516	60	0001	XXXX	RAU		

*Next instruction located at 0516 (Branch)

0500	69	0010	0513	LDD	0,0,0,0,0,0,0,7,8,9	+
0513	91	0516	0518	* BD1	0,0,0,0,0,0,0,7,8,9	+
0518	92	0521	0523	** BD2	0,0,0,0,0,0,0,7,8,9	+
0523	93	0526	0528	** BD3	0,0,0,0,0,0,0,7,8,9	+
0521	60	0002	XXXX	RAU		

*Next instruction located at 0518 (No Branch)
**Next instruction located at 0521 (Branch)

0500	69	0010	0513	LDD	0,0,0,0,0,0,0,7,9,9	-
0513	91	0516	0518	* BD1	0,0,0,0,0,0,0,7,9,9	-
0518	92	0521	0523	* BD2	0,0,0,0,0,0,0,7,9,9	-
0523	93	0526	0528	** BD3	0,0,0,0,0,0,0,7,9,9	-

*No Branch
**Error Stop

Figure 30. Branch Distributor Operation

Read Codes. The read codes (70RD1, 72RC1, 73RD2, 75RC3, 76RD3, 78RC3) also have a branching function. When a read code is executed, the question asked is "Was the LOAD hub on the control panel impulsed to indicate that the information now in the synchronizer entered from a LOAD card?" If the LOAD hub was impulsed, the location of the next instruction is specified by the D-address of the read instruction. If the LOAD hub was not impulsed, the location of the next instruction is specified by the I-address.

Miscellaneous Codes

00 NOP (No Operation). This code performs no operation. The D-address is bypassed, and the machine automatically refers to the location specified by the I-address. The D-address must be valid to prevent a storage selection error.

01 HLT (Stop-Halt). This code is conditioned by the PROGRAMMED switch on the control console. When the switch is set to STOP, this code stops program execution. When the switch is set to RUN, this code is treated like the NOP code.

69 LDD (Load Distributor). This code causes the contents of the general storage location specified by the D-address to be placed in the distributor. This code, in conjunction with the 24 STD code is used to transfer information from one general storage location to another. Figure 31 illustrates the use of these codes.

Table Lookup

The table lookup feature of the IBM 650 Data Processing System permits a table reference to be performed in a minimum time and with a minimum programming effort. The single instruction used in table lookup tells the machine to make a sequential search of a table. This search will continue until the desired item in the table is found, or some other condition indicated by the programmer has occurred. This eliminates the need for the many programmed comparisons that would otherwise be necessary to find any given item in a table.

Before beginning with the detailed description of the table lookup feature the following terminology should be understood:

TABLE	an arrangement in condensed form for ready reference of statistics, measures, etc.
ARGUMENT	the known reference factor necessary to find a desired item in a table.
FUNCTION	the unknown factor or factors within a table associated with the known reference factor (argument).

A table, therefore, will consist of a series of arguments (reference factors). Associated with each argument will be one or more functions (data within the table).

Table arguments are stored on the drum in ascending sequence by their absolute value (argument signs are ignored). Arguments may be a maximum of 10

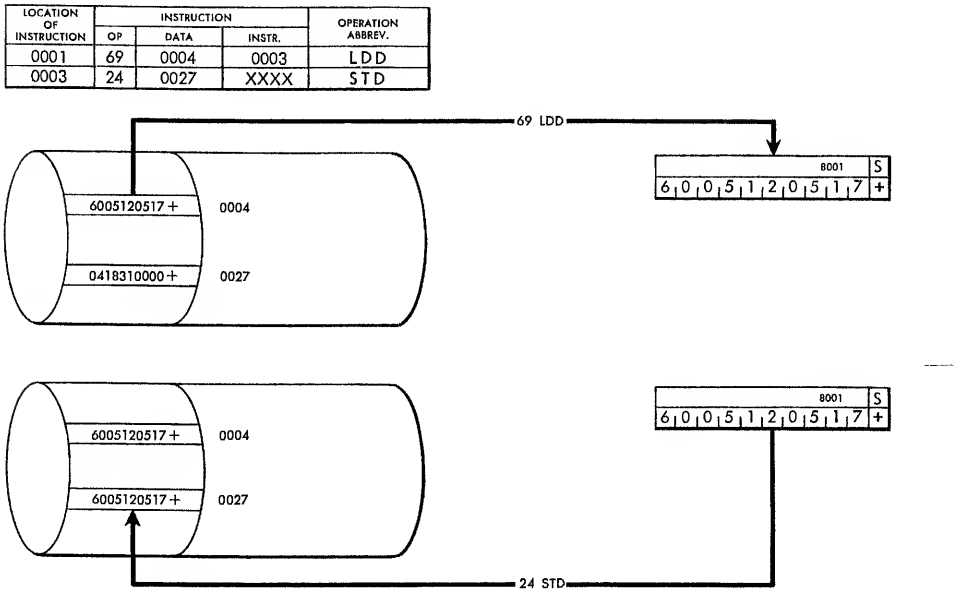


Figure 31. Relocating Information in Storage

digits in size and are stored, 48 in each band of general storage, except for the last band of a table, which may contain fewer. The last two storage locations in each band (0048-49; 0098-99; 0148-49; etc.) cannot be used to store table arguments; they may be used to store functions, instructions, etc. Table arguments must be stored in successive drum locations starting with the first word in a band (0000, 0050, 0100, etc.).

The table lookup operation will cause a search of the table to be made beginning with the first table location. The searching argument (in the distributor) will be compared against each of the table arguments until a table argument is found which is either equal-to or higher-than the search argument. Therefore, a table does not need to contain table arguments for all of the possible search arguments that will be encountered. When a search argument is not in the table, the search will stop on the next higher table argument. Interpolation can be accomplished by programming.

The fact that the search stops on a next higher number, as well as an equal, makes it possible to have the table argument and the associated function located in the same word. If this is done, the argument must be stored in the most significant positions of the word (to the left).

Functions may also be stored a fixed number of locations away from the argument. Thus, having found the location (N) of the argument, the function is located at N + C, (where C is fixed separation factor of the functions from the arguments).

The table lookup operation code (84 TLU) indicates to the machine that the TLU search is to be made. The band in which the search is to begin is indicated by the D-address of the TLU instruction. The search always begins at the first word of the band (0000, 0050, 0100, etc.). Therefore, if the instruction 84-0000 is given, the search begins at location 0000 of general storage. Likewise, if the instruction 84-0023 is given, the search still begins at location 0000.

Once the table search has been initiated, a count is kept of the number of drum locations that are passed without finding the argument that is being searched for. When the desired table argument is found, the count of locations passed is added to the D-address of the TLU instruction. The result of this addition is then placed in the lower accumulator, positions 8-5. This placement is automatic and is the end result of the TLU operation.

In Figure 32A the argument and function are in the same storage location. Note that the instruction given is 84-0000. The correct argument is found in location

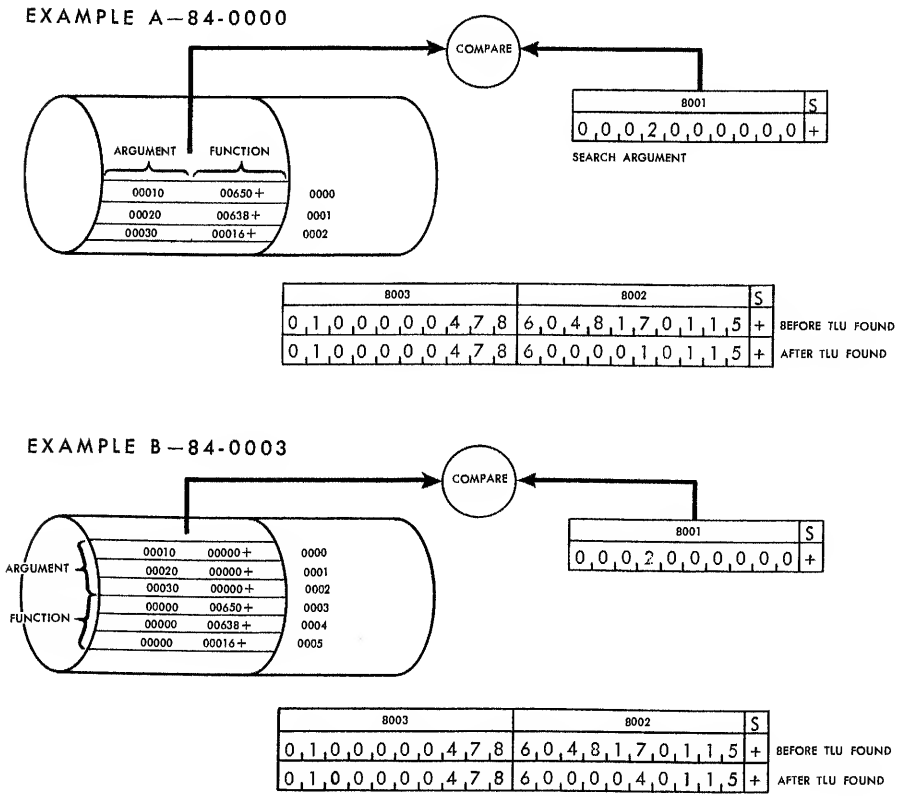


Figure 32. Generalized TLU Operation

0001 (one location has been passed before the correct location was found). Therefore, 1 is added to the original D-address of 0000, and the result (0001) is placed in positions 8-5 of the lower accumulator. In Figure 32B the argument and function are not in the same storage location. Rather, the functions are located in the three storage locations immediately after the arguments. In this case, the instruction given is 84-0003. The search still begins at location 0000, and the correct argument is still located at 0001. Because one location has been passed before the correct argument is found, 1 is added to the D-address (0003); and the result (0004) is placed in the lower accumulator, positions 8-5. This is the correct location of the table function.

It is convenient to use this method to handle short tables where the size of the argument or the function makes it impossible to locate them both in the same storage location. This method is applicable only where the number of table arguments is 48 or less. Also, it is possible to store several short tables in one band, provided proper placement of the argument will cause the search to be effective only in the applicable section of the band. When this method is used, it is necessary for the first argument of each table to be greater in value than the last argument of the previous table.

General Operation

Before the 84 TLU instruction is executed, the known search argument must be placed in the distributor. This is usually done with a 69 LDD instruction. The general sequence of steps in a TLU operation is:

1. Load distributor with known search argument.
2. Issue table lookup instruction. The D-address of this instruction determines in which band the search will begin.
3. Starting with the first word of the selected band, begin the automatic comparison between the contents of the storage location and the distributor. This search and comparison will continue until the argument in the storage location is either equal-to or higher-than the known search argument in the distributor.
4. A count is kept of the number of locations passed. This is done automatically by the machine. (If the table arguments exceed 48, it will be necessary to use more than one band of general storage. In this case the first 48 table arguments are placed in the first selected band and the overflow go on the next higher adjacent band).
5. If the machine goes through the first band without locating the desired argument, the search and comparison is automatically interrupted at the end of the 48th table argument. During the remaining two words of this band, the D-address of the TLU instruction is modified by the factor of 50, and the search is moved to the next higher band. The search continues with the first word of the new band.
6. When an equal or higher condition occurs, the number of words passed in the band is added to the D-address of the TLU instruction. The result of this addition is placed in positions 8-5 of the lower accumulator. This is done on a direct replacement basis. The other positions of the accumulator are not affected. At the completion of the TLU operation, the search argument is still in the distributor.
7. If the table argument and function are not in the same storage location, some modification of the contents of the accumulator is necessary (except for the example previously noted).

It is important that an *end-of-table—end-of-search* indication be incorporated into the table. Several methods of indicating end-of-search are possible. The most frequently used is to insert an argument of all 9's as the last argument of the table, to terminate TLU.

If no end-of-table indication is provided, a search argument larger than any of those in the table will cause the machine to continue searching until it encounters a location that contains a number equal-to or higher-than the search argument. Under this condition, the address of the location on which the search stopped would be used as the address obtained from the TLU operation. If no equal or greater number is encountered, the machine stops when it attempts to search location 2000 (invalid address).

If an end-of-table indication is undesirable, it is possible to compare the known argument with the last table argument. This way, it can be assured that the known argument falls within the range of the table.

On the TLU operation, the table arguments are validity checked. If any word in the table is invalid, the TLU operation stops and the address of the invalid word appears in the address lights on the console.

84 TLU (Table Lookup). This code performs an automatic table lookup using the D-address as the location of the band in which to begin the search. The I-address is used to locate the next instruction of the program. The argument for which the search is to be made must be in the distributor before this command is given. The address of the table argument equal-to or higher-than the search argument is placed in positions 8-5 of the lower accumulator. The search argument remains, unaltered, in the distributor.

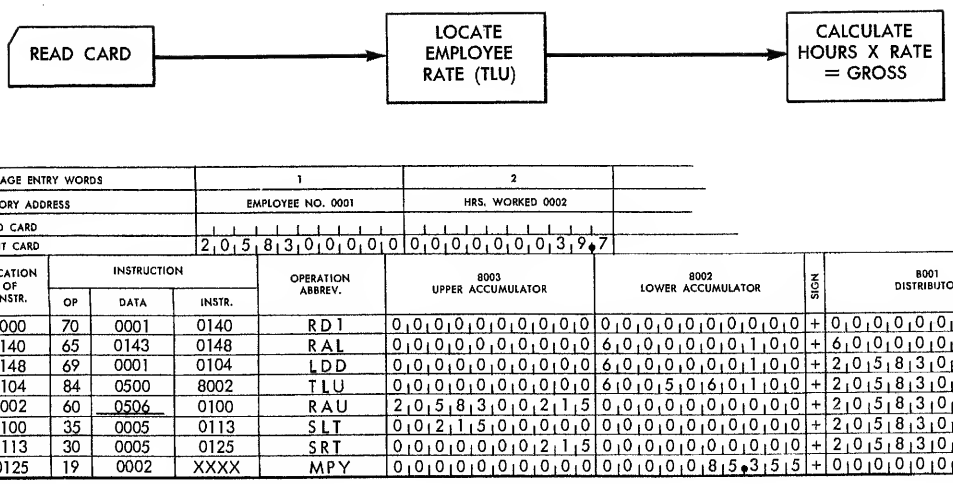


Figure 33. TLU Program Illustration

Program Example

Figure 33 is a sample block diagram and program using TLU. It is presented to show how TLU can be used in combination with other codes. The purpose of the program is to compute each employee's gross earnings from the information punched in his time card and information located by the TLU. The formula is:

HOURS $\times$ RATE = GROSS

Total-hours-worked is punched in the employee's time card. The rate is found by a TLU operation on employee number. As each card is read, the employee number (5 digits) is entered into the high-order positions (10-6) of word 0001, and the total-hours-worked (3 digits) is entered into the low-order positions of word 0002.

The table is stored on the drum beginning at word 0500. Each table word contains both the table argument (employee number) and associated function (rate). Figure 34 shows the first 10 table words. In addition to the table, program, and data, a skeleton instruction (60 0000 0100) is stored in word 0143.

DRUM TABLE

DRUM WORD	EMPLOYEE NUMBER (ARGUMENT)	PAY RATE (FUNCTION)
0500	02176	00 185
0501	03841	00 157
0502	17238	00 240
0503	17239	00 225
0504	18448	00 200
0505	20011	00 225
0506	20583	00 215
0507	21456	00 225
0508	34112	00 210
0509	35489	00 160

Figure 34. Drum Table

PROGRAM ANALYSIS

Program Step 1 — the data from the employee time card is read into storage.

Program Step 2 — the skeleton instruction is placed in the lower accumulator.

Program Step 3 — the search argument is loaded into the distributor.

Program Step 4 — the table search is begun. As a result of finding the item in location 0506. the skeleton instruction is modified to 60 0506 0100.

Program Step 5 — use the modified instruction to place the contents of the found location in the upper accumulator.

Program Step 6 — shift the accumulator to the left to lose employee number.

Program Step 7 — shift the accumulator to the right to reposition rate.

Program Step 8 — multiply hours-worked by the rate to determine gross. At the end of this step the gross wages are in the lower accumulator. The direction taken by the program would now depend upon that part of the block diagram which would follow the example. The next step could be (31 0001 xxxx) which would round the gross wages to the nearest penny.

Other Considerations

Before a TLU operation is performed, careful consideration must be given to the application itself. In some cases optimum results can be obtained from other programming techniques.

One such technique is where the search argument is a 4-digit number that falls in a block between the limits of 0000-1999. In this case the search argument

can be used as a direct address to locate the function.

Another area for consideration is where the table arguments may have the same absolute value but different signs. Because signs are not considered during TLU, two separate tables would probably be used. Where signs are a consideration it is possible that only one set of arguments would be stored and two different function locations used. The correct function location is then determined by the sign of the search argument.

Absolute Arithmetic Codes

The absolute arithmetic codes enable the absolute value (magnitude) of the incoming factor to be added or subtracted from the accumulator. During sign analysis, the sign of the value in the distributor is ignored and assumed to be positive. The absolute value in the distributor is always added or subtracted in the lower half of the accumulator.

17 AML (Add Magnitude to Lower). This code causes the contents of the D-address location to be added to the contents of the lower half of the accumulator. The sign of the factor in the distributor is always assumed to be positive. When the operation is completed, the distributor contains the D-address factor with its actual sign. Figure 35 shows the results obtained when using the 17 AML code.

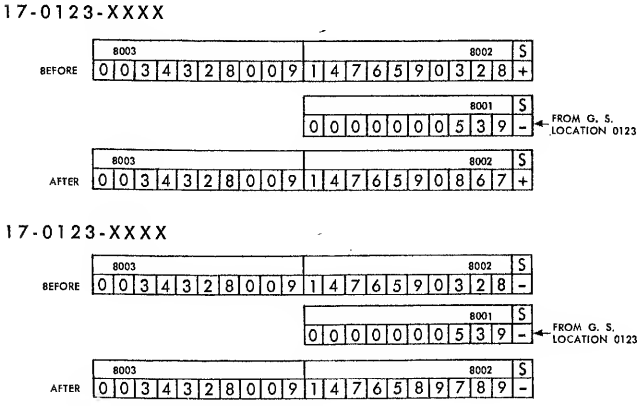


Figure 35. 17 AML Operation

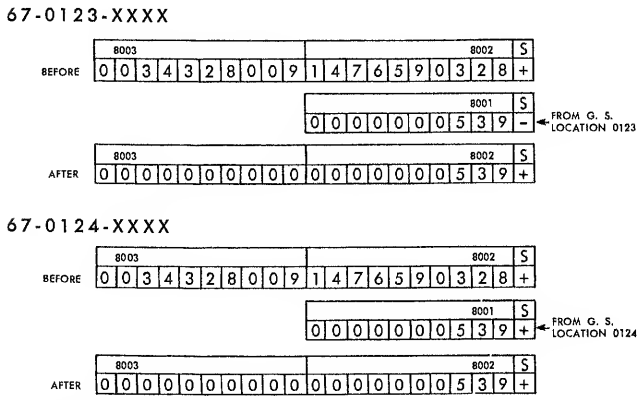


Figure 36. 67 RAM Operation

67 RAM (Reset and Add Magnitude to Lower). This code resets the entire accumulator to plus zero and adds the contents of the D-address location into the lower half of the accumulator as a positive factor regardless of its actual sign. When the operation is completed, the distributor contains the D-address factor with its actual sign. Figure 36 shows the results obtained when using the 67 RAM code.

18 SML (Subtract Magnitude from Lower). This code causes the contents of the D-address location to be subtracted from the contents of the lower half of the accumulator. The sign of this factor is always assumed to be positive. When the operation is completed, the distributor contains the D-address factor with its actual sign. Figure 37 shows the results obtained when using the 18 SML code.

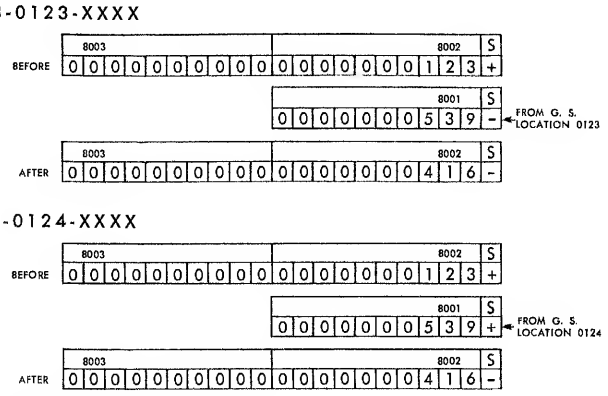


Figure 37. 18 SML Operation

68 RSM (Reset and Subtract Magnitude from Lower). This code resets the entire accumulator to plus zero and subtracts the contents of the D-address location into the lower half of the accumulator as a positive factor, regardless of its actual sign. When the operation is completed, the distributor contains the D-address factor with its actual sign. Figure 38 shows the results obtained with using the 68 RSM code.

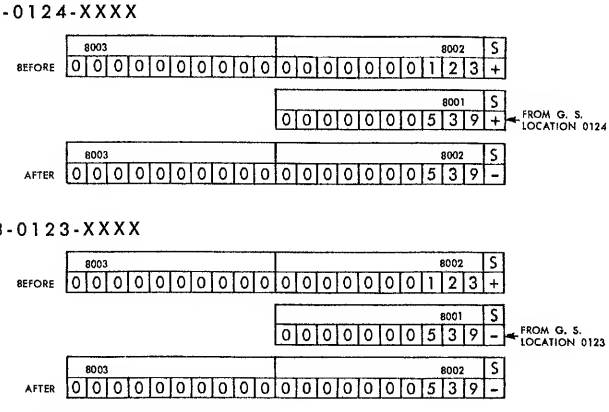


Figure 38. 68 RSM Operation

Optimum Programming

The 650 has been designed for high process speed and ease of programming. The programmer is not burdened with timing restrictions because internal interlocks are provided that prevent one instruction from interfering with another.

If the 650 is to operate efficiently, the data and instructions used in a problem must be so located in storage as to be immediately available to the drum reading heads when the program calls for them. That is, they must be in their *optimum* location. For example, (Figure 39) if an instruction calls for data from location B at the time the drum is passing the read heads, it is immediately available (optimum location). However, if the same instruction calls for data from location A, the drum must make almost a complete revolution before the data is available to the read heads. During this time, processing is stopped, waiting for the data to become available. Even though location C is not the optimum location, it is much better than location A. The same logic applies to instructions as they are called for during the course of program execution.

The non-productive waiting or searching time caused by poorly located data and instructions can be greater than the time it takes to actually execute the instruction. To realize the maximum efficiency of the system, the programmer should consult the Optimum Program Chart, Form X24-6219, which will guide him in the selection of the best storage locations for the data and instructions to be used in the problem.

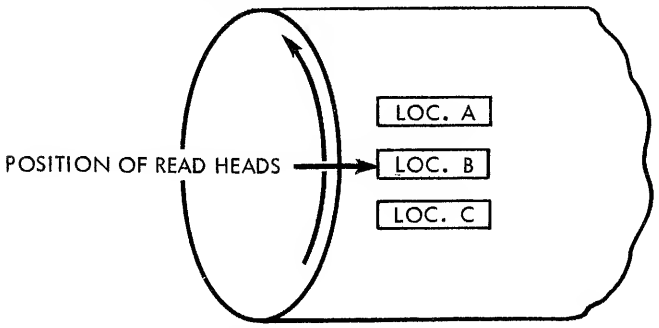


Figure 39. Locating Optimumly

Techniques

Any problem in which the speed of input or output is appreciably less than maximum because of lengthy process time, can be improved by optimum programming techniques. The amount of gain depends on the particular problem and the degree of optimizing applied by the programmer. Not every instruction can be in an optimum location, because occasionally conflict will occur between instructions. For example, data in an optimum location for a store operation may not be in an optimum location for a later add operation. Further, an instruction that is preceded by several branch instructions cannot easily be in the optimum location for each of these branch instructions.

Once it has been determined that optimum programming will be used, these techniques can be followed:

1. The program can be optimized as it is written. Normally, with this procedure, locations chosen at the beginning of the problem are poorly located for processing in a later section of the problem.
   This technique lends itself to rapid programming and is almost as fast as programming sequentially. Even though the latter sections of the problem are not completely optimized, the degree of optimization will result in a significant increase in over-all speed. In this respect, this type of optimization justifies the small amount of additional effort required.
2. The second method of optimizing requires a certain amount of planning before programming. Instead of optimizing each program step as it is written, the programmer must think ahead to visualize the possible effect on later steps that will use the same data, or perhaps branch to the same instruction.
   Many possibilities exist when programming this way. After the program has been completed, a simple re-arrangement at the beginning could possibly improve a later portion of the problem. It may be necessary to re-program the problem several times to achieve maximum machine efficiency. Obviously, such a procedure requires more time than sequential programming or method 1.
3. Another method uses interpretive and assembly routines such as Symbolic Optimum Assembly Program (SOAP). This system is described in Form 32-7646.

The second method of optimizing should be used only when method 1 does not produce maximum input or output. The problems justifying this method of optimizing are those which are run frequently and in which the small time saving on every run is significant over an extended period. It is not always necessary to program the entire problem this way, but only those segments or subroutines that are most frequently used.

To establish rules of optimum programming, each operation has been analyzed to determine the number of word-times required for its analysis and execution. In most cases, it is not practical to optimum-program such operations as multiplication, division, and table

lookup, because of the wide variation of factors.

From a timing standpoint, there are 40 drum locations (one per band) that are time-equivalent. Words 0001, 0051, 0101, etc., all pass the read heads at the same time. Similarly, words 0002, 0052, 0102, etc., all pass the read heads at the same time. Thus, if optimum programming indicates a location of 0005, the information can be placed in 0005, 0055, 0105, etc., without changing the time required for the operation. Once an optimum location has been determined, there are 39 additional locations that are equally accessible. Another factor of importance is that the I-address of many instructions can be optimum within a range of locations. This is because the 650 can overlap arithmetical processing with the search for the next instruction.

Accumulator Considerations

Because the 650 uses a 20-position accumulator and because the entire accumulator is involved in arithmetic and shifting operations, each half of the accumulator is synchronized to specific word times. The lower half (8002) is synchronized to *even* words. The upper half (8003) is synchronized to *odd* words. This means, when any arithmetic or shifting operation is to be performed it is necessary to wait for an *even* word time before the operation can begin. This is why the optimum program chart may show differences in timing for odd-even locations. Similarly, store-accumulator operations must wait for an *even* word when the lower is to be stored, and an *odd* word when the upper is to be stored. Likewise any operation that involves the accumulator may be affected by these considerations. The optimum program chart covers all possible conditions.

Input-Output Interlocks

Interlocks are provided in the input-output units to assure that when the program calls for any input and/or output (Op codes 70-78), the mechanical operation is begun before program execution is allowed to advance to the I-address of the input-output instruction. Because these interlocks are on mechanical devices, the I-addresses of the associated program instructions need not be optimized. The exact amount of time that program execution is stopped depends on:

1. the amount of process time exclusive of input-output commands
2. the specific input-output unit involved (533, 537, 407)
3. the operation (RD, WR, RC/WR) called for

533 Interlock Time

Each RD or WR instruction for the 533 results in an average of 61 ms interlock time. This is true only if the section (input or output) receiving the command is operating at less than its maximum rate.

537 Interlock Time

Each WR instruction for the 537 results in an average of 60 ms interlock time. Each RD or combination RC/WR instruction results in an average of 90 ms interlock time. In each case the 537 is assumed to be operating at less than maximum rate.

407 Interlock Time

To determine the average interlock time for the 407 it is necessary to know the amount of computing time. When the compute time is known, 650 Bulletin 6, (Form 32-7990) can be used to determine average interlock time.

Available Computing Time

If the input-output units are to run at maximum speed, the computing time between each input-output command cannot exceed certain maximums that are set by the speed of the input-output device being used.

Any input-output cycle, once started, cannot be repeated until the cycle in progress has been completed. The cycle times for the various input-output units are:

1. 533 input — 300 ms
2. 533 output — 600 ms
3. 537 input-output — 390 ms (approximate)
4. 407 input — output — 400 ms

To determine the compute time available during any one cycle, the interlock time of the unit is subtracted from the total time available during the cycle.

For example, assume a system using a 533 for input-output, and an application that requires one input and one output card for each transaction. The maximum possible speed is 100 transactions per minute, determined by the output of the 533. This in turn sets a maximum of 600 ms for each transaction. To determine the available compute time, the interlock times (61 ms for input + 61 ms for output) are subtracted from the total available time. This gives 478 ms as the available compute time.

How many instructions can be executed in 478 ms? This is determined primarily by the degree of optimization used by the programmer. Studies have shown that a well-optimized program will execute about 3 instructions per drum revolution. In 478 ms the drum will make about 100 revolutions (4.8 ms per revolution), which indicates that about 300 instructions can be executed and still maintain maximum output. The same principles can be applied to any input-output unit or combination of units. The over-all speed will be set by the speed of the slowest unit kept running at maximum rate.

IBM OPTIMUM PROGRAM CHART—650 DATA PROCESSING SYSTEM

	Location of Inst. n, n_e	Operation	Data		Instruction
ARITHMETIC CODES	n	Add-Subtract:10, 11,15,16,17,18,60, 61,65,66,67,68...	n+3	EVEN	d+5
				ODD	d+4
	n	MPY(19)	n+3	EVEN	d+21+2Σm
				ODD	d+20+2Σm
	n	DIV(14),DVR(64)	n+3	EVEN	d+61+2Σq
				ODD	d+60+2Σq
BRANCHING CODES	EVEN ODD	NZU(44)	n+3 n+4		n+4 n+5
	EVEN ODD	NZE(45)	n+4 n+3		n+5 n+4
	n	BMI(46)	n+3		n+4
	n	BOV(47)	n+3		n+5
	n	BD0(90), BD9(99)	n+4		n+5
	n	BD1-BD8 (91-98)	n+3		n+5
	n	NTS(25), NEF(54)	n+4		n+5
	n	NZA-B-C (40,42,48) BMA-B-C (41,43,49)	n+3		n+4
	n	BIN(26)			n+4
MISCELLANEOUS OPERATION CODES	n	LDD(69), STD(24)	n+3		d+3
	n	TLU(84)	n+3	EVEN	A+5
				ODD	A+6
	n	NOP(00), HLT(01)			n+4
	EVEN ODD	STL(20)	n+5 n+4		d+3
	EVEN ODD	STU(21)	n+4 n+5		d+3
	EVEN ODD	SDA(22), SIA(23)	n+3 n+4		d+3
	EVEN ODD	SRT(30), SLT(35), SCT(36), S=0			n+6 n+5
	EVEN ODD	SRT(30), SLT(35), SCT(36), S=1,2			n+7 n+6
	EVEN ODD	SRT(30), SLT(35), SCT(36), S=3-9			n+7 thru (2S+3) n+6 thru (2S+2)
	EVEN ODD	SCT(36) S=10	**		n+7 thru (2S+3) n+6 thru (2S+2)
	EVEN ODD	SRD(31) S=1-10			n+7 thru (2S+5) n+6 thru (2S+4)
INDEXING REG. CODES	n	Add-Subtract - 50, 51,52,53,58,59,80, 81,82,83,88,89	0000-1999		n+5*
	n	"	9000-9059		n+7*
	n	"	8000		n+7*
	n	"	8001		n+5*
	EVEN ODD	"	8002		n+8* n+7*
	EVEN ODD	"	8003		n+7* n+8*
	n	NZA-B-C (40,42,48) BMA-B-C (41,43,49)	n+3		n+4

	Location of Inst. n, n_e	Operation	Data	Instruction
USING 800X ADDRESSES★	EVEN ODD	All Add-Subtract (Accumulators)	8000, 8003 8005,8006,8007	n+7 n+8
	EVEN ODD	"	8001	n+7 n+6
	EVEN ODD	"	8002	n+9 n+8
	n	LDD(69)	8000, 8005-6-7	n+6
	EVEN ODD	LDD(69)	8002	n+7 n+6
	EVEN ODD	LDD(69)	8003	n+6 n+7
		All Add-Subtract (Accumulators)	EVEN ODD	(8000,8001, 8003) d+5# d+4#
		"	EVEN ODD	(8002) d+6# d+5#
		"	EVEN ODD	(8005-6-7) I_1=d+9# I_1=d+8#
		LDD(69)	d	(8000,8001) d+3
		"	EVEN ODD	(8002) d+4 d+3
		"	EVEN ODD	(8003) d+3 d+4
		"	d	(8005-6-7) I_1=d+7
IAS	n	LIB(08), LDI(09) SIB(28), STI(29)	n+3	d+W+2
	n	SET(27)		n+5
TAPE	n	RTN(04), RTA(05), WTN(06), WTA(07), RTC(03), RWD(55), WTM(56), BST(57)		n+5
	n	NTS(25), NEF(54)	n+4	n+5
DISK STOR.	n	SDS(85), RDS(86), WDS(87)		n+5
	n	BIN(26)		n+4
FLOATING DECIMAL	n	UFA(02)	n+3	EVEN d+5 thru 33** ODD d+4 thru 32**
	n	FAD(32), FSB(33) FAM(37), FSM(38)	n+3	EVEN d+5 thru 59** ODD d+4 thru 58**
	n	FMP(39)	n+3	EVEN d+5 thru 47** ODD d+4 thru 46**
	n	FDV(34)	n+3	EVEN d+5 thru 41** ODD d+4 thru 40**

ADDR. MOD. IND. REG.

When optimizing the addresses of a "tagged" instruction the programmer must mentally adjust the Location of Instruction (n) to an Effective Location of Instruction (n_e) before adding the optimizing factors. The optimizing of a tagged instruction is always done from the effective location (n_e) and this can change the even-odd word consideration. With one address tagged, (n_e) = n + 1. With both addresses tagged, (n_e) = n + 2.

Σm: Sum of Multiplier Digits
Σq: Sum of Quotient Digits
S: Number of Shifts
W: Number of words moved

NOTATIONS:
A: Location of found argument (TLU)
d: Data Address of Instruction
i: Instruction Address of Instruction

*Add one to the "I" Address if o complement cycle is taken.
**Where o Low-High limit is indicated for an "I" Address, the following instruction should be optimized from the High limit. This applies only when the "I" Address is placed between the specified limits.
Add two if o complement cycle is taken.
I_1 Instruction address specified by the Indexing Register. The times shown include the execution of the NOP(00) instruction which follows on "I" Address of 8005-6-7.
★ The access time to 90XX locations is always equivalent to an optimum drum location if IAS is not interlocked.

Figure 40. Optimum Program Chart

Using The Optimum Program Chart

The Optimum Program Chart (Figure 40) is a tool to aid the programmer. Its format is similar to the 650 Planning Chart, Form X24-6151 and 650 Program Sheet Form X24-6181. It is divided into functional groups. All of the arithmetic codes are in one section. All of the branching codes are in another section and the miscellaneous codes (shifting, storing, etc.) are in another section. Because 800X addresses require special consideration when used as sources of data or instructions, they are set aside in a special section.

Arithmetic Codes

To illustrate the use of the arithmetic section of the optimum program chart, the hypothetical problem in Figure 41 will be programmed. Factors A and B are entered from the input card. Factors C and D are constants that are stored in the machine during program loading. Figure 41 also shows the skeleton and completed programs.

The I-address of the RD1 instruction is made 0048 so that factor A, entered into 0001 from the card, will be in an optimum location. Location 0048 was chosen by using the section of the chart on arithmetic codes and by working backward from the fixed location 0001. The chart shows that the optimum D-address of an add code is 3 words away from where the instruction is located in storage (n + 3). Because the D-address location is fixed in this case, the location of the instruction (n) is optimized by subtracting 3 from the D-address. The I-address of the RD command is not optimized with respect to its D-address because the input-output interlock, described previously, acts between these two addresses.

To select the optimum I-address for the 60 RAU instruction, the chart shows (in the section labeled Arithmetic Codes) that 5 is added to the D-address when it is an EVEN numbered location and 4 is added when the D-address is an ODD numbered location. This makes the optimum I-address 0055. Location 0055 is chosen rather than 0005 because 0005 is in the read-in area and any information in this location is lost on a read-in cycle.

When the I-address of the 60 RAU instruction is chosen, it becomes the location of instruction (n) of the 10 AUP command. The D-address for the 10 AUP command can now be set. The arithmetic codes section of the chart shows that the optimum location is 3 words away from the location of instruction. This makes the optimum D-address 0008. Note that this is still within the read-in area and factor B can easily be entered into 0008 using control-panel wiring. In this way, the control panel actively assists the programmer in achieving program efficiency. For this reason, control panel planning should go hand in hand with programming.

After the D-address of the 10 AUP command is set, the I-address can be chosen by using the chart. Because the D-address is an EVEN location the I-address is 5 words away from the D-address. This makes the optimum I-address 0013.

The D-address of the 11 SUP command can now be optimized and will be 0016, which is n (0013) + 3 as shown by the chart. This is factor C and it is loaded into the machine at the same time the program is loaded.

The I-address of the 11 SUP command is placed 5 words away from the D-address because the data is in an EVEN location. NOTE: if a conflict arose later for location 0016, the D-address of the 11 SUP instruction could be changed to location 0017 without changing the I-address or affecting the operating speed.

The D-address of the 19 MPY command is placed 3 words away from the instruction location as indicated by the chart. Unless the exact value of the multiplier factor (upper accumulator) is known, the I-address cannot be optimized.

Note that the optimum D-address of all arithmetic codes is 3 words away from the location of the instruction (n + 3). During this time the instruction is analyzed to determine what is to be done and the necessary circuitry is set up to read the data into the distributor. If the data is in an optimum location it will immediately enter the distributor. If it is not in an optimum location, time is lost while the drum moves to the data location. Once the data has entered

PROBLEM: (A+B−C)×(D)=E

Location of Instruction	Instruction			Operation Abbrev.	
	OP	Data	Instruction		
8000	70	0001		RD1	
	60	0001		RAU	+A
	10			AUP	+B
	11			SUP	−C
	19		XXXX	MPY	×D

Location of Instruction	Instruction			Operation Abbrev.
	OP	Data	Instruction	
8000	70	0001	0048	RD1
0048	60	0001	0055	RAU
0055	10	0008	0013	AUP
0013	11	0016	0021	SUP
0021	19	0024	XXXX	MPY

Figure 41. Optimizing Arithmetic Codes

the distributor, the arithmetic operation will proceed while the I-address of the instruction is analyzed and the next instruction is entered from the drum into the program register. In this way, as soon as one operation is completed, the analysis of the next will begin. No time is lost searching for the instruction when the optimum location is used.

Branching Codes

These codes make tests to determine the location of the next instruction to be executed. In all cases, faster program execution will result when branching logic is designed so that a majority of the branches are taken from D-addresses. In other words, design program logic so that the branching questions are being answered YES most of the time.

When a series of branch instructions are being used, they should be arranged in such a way that the most frequently encountered item is tested first. For example, an inventory application could have three types of input cards: receipts, on-order and issues. To determine what process routine to use, a series of branch distributor instructions could be set up. The sequence of tests would depend on the relative number of cards of each type that will be encountered. If issues are the largest quantity followed by receipts and on-order cards, program logic is best designed as shown in Figure 42.

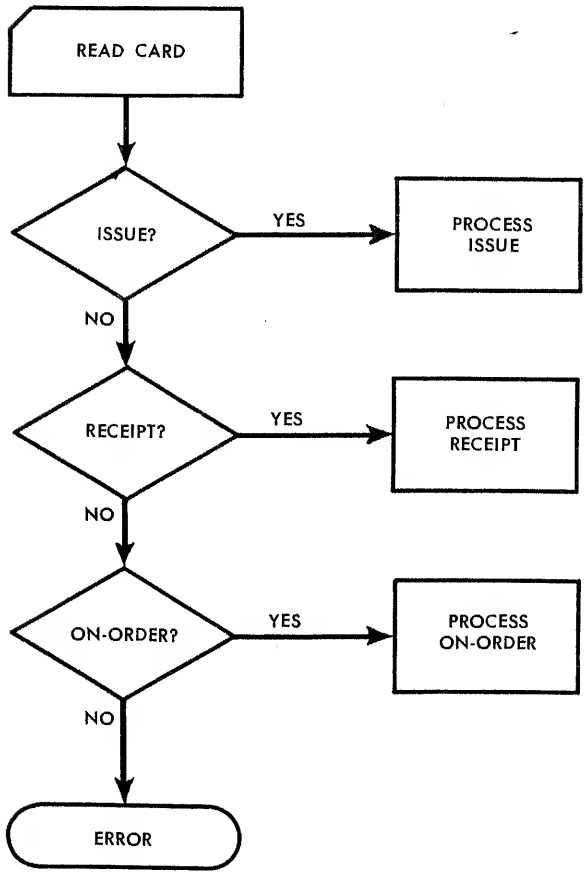


Figure 42. Branch Code Logic

Another point to consider is the effect of EVEN-ODD location on the execute time of the 44 NZU and 45 NZE operations. Note that the EVEN-ODD consideration for one is the opposite of the other. There may be times when either operation will give the desired test results. In this case, choose the one that will give the lowest operating time depending on whether n is EVEN or ODD.

Shifting Codes

The time it takes to execute a shift operation depends on the number of positions the accumulator is to be shifted. Because the time is variable and because only the accumulator is involved in the shift, the I-address of the shift instruction can be optimum within a range of locations that gives the programmer a degree of flexibility when conflict for locations arise. While the shifting operation is being performed, the 650 can enter the next instruction into the program register. However, for programming purposes, the high limit is used for the *n* of the next instruction.

Example: The I-address of the following shift instruction is to be optimized as is the D-address of the following instruction:

n	OP	DATA	INST	
0117	30	0007	XXXX	SRT
	15	XXXX		ALO

When the formula shown on the optimum program chart is applied, the I-address low limit is 0123 (n + 6) and the high limit is 0133 (2S + 2). The I-address can be placed anywhere between these two limits. When the 15 ALO D-address is optimized, the high limit (0133) must be used as n. This is true even if the I-address of the 30 SRT command is set at 0123. Therefore, the optimum D-address for the 15 ALO command is 0136.

In many instances, the programmer can use the control panel of the input-output units to shift information as it is being entered into or taken out of the system. Whenever this is possible, it should be done, because it reduces program execution time by eliminating a shift instruction.

800X Addresses

Because the storage-entry switches (8000), distributor (8001), lower accumulator (8002), and upper accumulator (8003), are addressable, they may be used as D- and I-addresses in many instructions. When an 800X address is so used, special rules apply for integrating these instructions into the program that is being optimized.

8000. Because 8000 is immediately accessible, any timing associated with its use is the same as an optimum drum location, except when it is the I-address of certain add-subtract instructions. Also, when used as the I-address a shift instruction, it is equivalent to the high limit.

Example: Optimize the I-address of the following instruction:

n	OP	DATA	INST	
1587	10	8000	XXXX	AUP

The chart indicates that the optimum I-address in this case is $n + 8$ because n (1587) is ODD. This makes 1595 the optimum I-address. In this case 8000 is equivalent to 1590 (n + 3). Because this is an EVEN word time, the I-address is d + 5 or 1595.

Examples: Optimize the D-address of the second instructions:

n	OP	DATA	INST	
0001	24	0004	8000	STD
8000	10	XXXX		AUP

The chart shows that 8000 is the *equivalent* of d + 3 when it is the I-address of a 24 STD instruction. Therefore, the effective *n* of the 10 AUP command is 0007 and the optimum D-address is 0010.

n	OP	DATA	INST	
0001	11	0004	8000	SUP
8000	10	XXXX		AUP

The chart shows that 8000 is the equivalent of $d + 5$ (d-EVEN) unless a complement cycle is taken, in which case it would be $d + 7$. An accumulator complement cycle can only occur if a 10 AUP, 11 SUP, 15 ALO, 16 SLO or 18 SML operation results in a change of accumulator sign from plus to minus. The additional two word times are used to change the results in the accumulator from a complement amount to a true amount with a minus sign. Therefore, the effective location of the second instruction is either 0009 or 0011 depending on the factors involved. This makes the D-address optimum at either 0012 or 0014.

n	OP	DATA	INST	
0001	69	8000	8000	LDD
8000	10	XXXX		AUP

The equivalent drum word for the I-address of the 69 LDD command is 0007 (n + 6) as shown by the chart. This in turn is the effective n of the 10 AUP command. Therefore, the optimum D-address is 0010.

8001. The distributor, like the storage entry switches (8000), is immediately accessible. Therefore, the times associated with an 8001 address are the same as 8000 except when it is used as the D-address of an arithmetic operation, in which case the time may be less.

Example: Optimize the I-address:

n	OP	DATA	INST	
1587	10	8001	XXXX	AUP

The chart shows that the optimum I-address is $n + 6$ when n is ODD. Therefore, the optimum I-address is 1593 which is two words faster than the same instruction with an 8000 D-address. The time is reduced because it is not necessary to move data to the distributor before beginning the add operation.

8002. The lower accumulator can only read out on EVEN word times. This means that in some cases an instruction with an 8002 address will take longer to execute than the same instruction with an optimized drum address.

Example: Optimize the I-address:

n	OP	DATA	INST	
1588	60	8002	XXXX	RAU

The chart shows that the optimum I-address in this case is n + 9 because n is EVEN. Therefore, the I-address is 1597.

Examples: Optimize the D-address of the second instructions:

n	OP	DATA	INST	
0001	11	0004	8002	SUP
8002	10	XXXX		AUP

The chart shows that 8002 is the equivalent of $d + 6$ unless a complement cycle is taken, in which case it would be $d + 8$. Therefore, the effective *n* of the second instruction is either 0010 or 0012. This places the D-address at 0013 or 0015.

n	OP	DATA	INST	
0001	65	8002	8002	RAL
8002	10	XXXX		AUP

The equivalent drum word for the I-address of the 65 RAL command is 0009 (n + 8). However, because this is an ODD word and 8002 is available only on an EVEN word, the equivalent location must move to 0010. This is then the effective *n* of the second instruction and, therefore, the optimum D-address would be 0013.

8003. The upper accumulator can only read out at ODD word times. This can have an effect similar to the EVEN word consideration for 8002.

Example: Optimize the D-address of the second instruction:

n	OP	DATA	INST	
0001	60	8003	8003	RAU
8003	15	XXXX		ALO

The equivalent drum word for the I-address of the 60 RAU command is 0009 (n + 8). Because this is an ODD word, 8003 can read out immediately. Therefore, 0009 is the effective *n* of the second instruction and the optimum D-address is 0012.

Programming From Fixed Locations

To illustrate another possible approach to optimum programming, the program shown in Figure 33 will be stripped of all addresses that are not fixed and then optimized from the chart. Figure 43 shows the skeleton program. The fixed locations are:

1. RD1 instruction – 8000
2. Employee number – 0001
3. Start of table search – 0500
4. I-address of TLU instruction – 8002
5. Hours worked – 0002

Location of Instruction	Instruction			Operation Abbrev.
	OP	Data	Instruction	
8000	70	0001		RD1
	65			RAL
	69	0001		LDD
	84	0500	8002	TLU
8002	60	XXXX		RAU
	35	0005		SLT
	30	0005		SRT
	19	0002	XXXX	MPY

Figure 43.

Some of the addresses cannot be effectively optimized because of the nature of the instruction of which they are part. These are:

1. I-address of the RD1 instruction because of the input-output interlock previously described. There is no way of accurately predicting the position of the drum when this interlock is removed.
2. I-address of the RAU (60) command. Because the TLU can stop anywhere within the table, the word time when the instruction from 8002 enters the program register is unknown as is the D-address of the RAU command.
3. I-address of the MPY (19) command because the execution time of this command is determined by the size of the multiplier factor.

Working within the framework of the fixed locations and those addresses that cannot be effectively optimized, the remaining locations can be chosen.

The first location to be chosen is that of the third instruction (69 LDD). This point is selected because the D-address is fixed. The optimum program chart shows that the best D-address of a 69 LDD code is $n + 3$. Because 0001 is fixed, it is only necessary to subtract 3 from 0001 to get the optimum location (0048) of the instruction. This in turn fixes the I-address of the previous instruction (65 RAL).

The next step is to continue working backward to choose the D-address of the 65 RAL instruction. The chart shows that the optimum location of the I-address is either $d + 5$ or $d + 4$ depending on whether the D-address is in an EVEN or ODD location. Therefore, the D-address will be 0043. An analysis here shows that the I-address of the 65 RAL instruction could be either 0047 or 0048 without affecting machine speed. However, by looking forward it can be seen that if employee number were shifted to 0002 on the input, a two word time difference occurs.

The next step is to determine the I-address of the RD1 instruction. Because we are still working backward, this will be determined by the D-address of the 65 RAL command. The chart shows the optimum D-address is $n + 3$. This gives 0040 as the best I-address for the RD1 instruction. Figure 44 shows the skeleton program with the preceding addresses added and what the program might look like if employee number were entered into 0002.

Location of Instruction	Instruction		
	OP	Data	Instruction
8000	70	0001	0040
0040	65	0043	0048
0048	69	0001	
	84	0500	8002
8002	60	XXXX	
	35	0005	
	30	0005	
	19	0002	

Location of Instruction	Instruction		
	OP	Data	Instruction
8000	70	0001	0042
0042	65	0045	0049
0049	69	0002	

Figure 44.

The next step is to optimize the I-address of the 69 LDD instruction. The I-address can be within a wide range of limits because the next instruction (TLU) has a fixed starting time. In this case the lower limit is set by the D-address of the 69 LDD command while the upper limit is set by the TLU command itself (all TLU operations start at the beginning of a band – 0000, 0050,

0100, etc.). The low limit is 0004 (d + 3) and the high limit is 0047. This allows the programmer a great deal of flexibility in later stages of the program. For purposes of illustration 0054 is used.

An examination of the rest of the program indicates that programming should now move to the next fixed location (0002). From this point optimizing proceeds backward until the previously addressed instructions are reached.

To determine the best location (n) for the 19 MPY command, 3 is subtracted from the fixed D-address. This places the 19 MPY command in location 0049. This in turn becomes the high limit for I-address of the 30 SRT command. The range of limits for a right shift of 5 is either 7-13 or 6-12 dependent on whether the command is in an EVEN or ODD location. The low limit of the I-address is then 0043, and the location of the 30 SRT command is 0037. If a conflict exists for location 0049, the I-address of the 30 SRT command can be shifted to 0047, 0046, etc., without affecting operating speed.

The range of limits for a left shift of 5 is the same as the right shift of 5. With 0037 chosen as the location of the 30 SRT command this in turn gives the high limit for the I-address of the 35 SLT command. The low limit is 0031. The 35 SLT instruction would be located at 0025. This in turn becomes the I-address of the 60 RAU command, which is entered into the system as a constant in location 0043 at the time the program is loaded.

This method of programming results in the fastest possible execution time that is consistent with problem limitations. Figure 45 shows the completed program.

Location of Instruction	Instruction			Operation Abbrev.
	OP	Data	Instruction	
8000	70	0001	0040	RD1
0040	65	0043	0048	RAL
0048	69	0001	0054	LDD
0054	84	0500	8002	TLU
8002	60	XXXX	0025	RAU
0025	35	0005	0037	SLT
0037	30	0005	0049	SRT
0049	19	0002	XXXX	MPY

Figure 45.

Sequence Chart (Figure 46)

The sequence chart shows the steps taken in the interpretation and execution of the various types of operations and the word times required for each step. In several cases the sequence chart branches into two parallel paths; this is to show that two functions are being performed simultaneously. One of the parallel branches represents the arithmetical process, and the other, the process of obtaining the next instruction.

Analyze the steps carried out by the machine in the performance of any operation; in every case certain fundamental word times are required. Begin consideration of each instruction at the time the instruction has been located. Starting at this point, the first word time of every operation is used to transfer the instruction from its memory location into the program register. The next word time is used to transfer the operation code to the operation register, and the data address, to the address register. The steps performed during the word times beyond this point depend upon the particular instruction being executed.

Consider, for example, an instruction calling for an add lower (15) operation. Assume that the add lower instruction is in location 0001. Then, as location 0001 passes the reading heads, the instruction is read into the program register (1 word time). As location 0002 passes the reading heads, the operation code (15) is transferred to the operation register, and the data address (XXXX) is transferred to the address register. As location 0003 passes the reading heads, the control circuits necessary to perform addition are set up, and the distributor is signaled to read in. When the location specified by the data address passes the reading heads, its contents will be read into the distributor. Therefore, if the data address of the add lower instruction had been 0004, the search time for the factor to be added would be zero. Thus, location 0004, or any of the 39 other locations having the same angular position, would be the optimum location for data in this example.

Assuming the data is in location 0004, the factor to be added enters the distributor as location 0004 passes the reading heads. The machine is now ready to add the factor in the distributor to the contents of the lower half of the accumulator. Because location 0005 is ready to pass the reading heads, the machine is starting an odd word time. The accumulator contents can only be read into the adder beginning with an even word time. For this reason, nothing is accomplished as location 0005 passes the reading heads. As location 0006 passes the reading heads, the contents of the distributor are added into the lower half of the accumulator. Simultaneously with the actual addition operation, a restart signal is given to the program

control system. During the time location 0007 is passing the reading heads, the contents of the upper half of the accumulator are passing through the adder, and the instruction address is transferred to the address register.

If complementing the accumulator contents is not necessary, the arithmetical interlock A is removed as location 0008 passes the reading heads, and the program register is readied to read in the next instruction. Therefore, if the instruction address of the add lower instruction had been 0009, the search time for the next instruction would be zero. Thus, location 0009, or any of the other 39 locations having the same angular position, would be the optimum location for the next instruction in this example.

If it is necessary to complement the accumulator contents, the over-all timing is not affected. Under this condition, the simultaneous operation continues while the next instruction is read into the program register, and the operation code and data address are transferred into the operation and address registers. Thus, the next instruction in process reaches the interlock point just as the interlock is removed.

The foregoing add operation and the location of the next instruction require eight word times for execution when optimum programming is used. The same two functions could require as much as 106 word times if the data and next instruction were placed in the worst possible locations. Of course, the factors would rarely fall in such poor relationship to each other; but when sequential locations for instructions are used, it can easily be seen that each operation and location of the next instruction must take at least 51 word times.

Referring to the sequence chart again, multiplication, division, and shifting operations may vary in the number of word times required for the arithmetical function. These operations offer latitude for positioning the succeeding instruction depending upon the time required for the multiply, divide, or shift function.

To continue with the example, assume that the instruction found in location 0009 calls for a shift right of three positions. The instruction is read into the program register as location 0009 passes the reading heads. During the time location 0010 passes the reading heads, the operation code and data address are transferred to the operation and address registers. Because no drum data location is used in a shift operation, the only concern of the programmer is in positioning the following instruction.

As location 0011 passes the reading heads, the shift control is readied. Accumulator read-out can begin immediately, because location 0012 (even word time) is ready to pass the reading heads. Parallel operations begin at this point. During word time 0012, actual shifting is begun, and the restart signal is given to the control unit. The next word time (0013) is used to complete a shift of one position and to transfer the instruction address to the address register. During word time 0014, the second position of shifting is begun, and the program register is signaled to receive the next instruction.

At this point, latitude for positioning the succeeding instruction is available. The next instruction could be in location 0015, which would be the lower limit for optimum location. However, the shifting operation is not completed until word 0017, and interlock A is not removed until 0018. Therefore, the instruction could be placed in location 0016 or 0017 and still have the new operation code and data address in the operation and address registers by the end of word time 0018 when the interlock is removed. Thus, location 0017 is the upper limit for optimum location of the instruction. Any of the three locations 0015, 0016, or 0017 gives zero access time for obtaining the instruction.

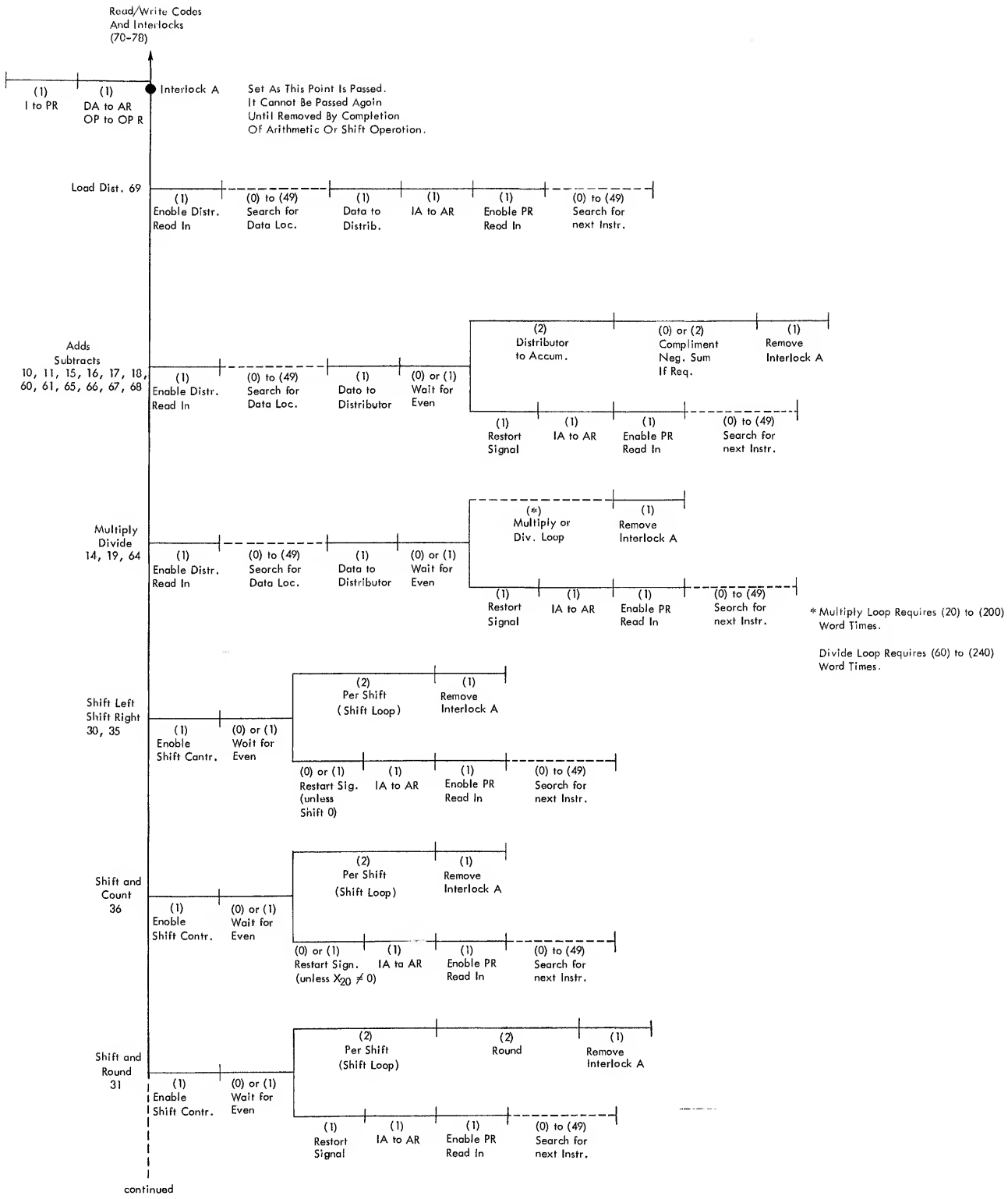

Read/Write Codes
And Interlocks
(70-78)
(1)
I to PR
(1)
DA to AR
OP to OP R
Interlock A
Set As This Point Is Passed.
It Cannot Be Passed Again
Until Removed By Completion
Of Arithmetic Or Shift Operation.
Load Dist. 69
(1)
Enable Distr.
Read In
(0) to (49)
Search for
Data Loc.
(1)
Data to
Distrib.
(1)
IA to AR
(1)
Enable PR
Read In
(0) to (49)
Search for
next Instr.
Adds
Subtracts
10, 11, 15, 16, 17, 18,
60, 61, 65, 66, 67, 68
(1)
Enable Distr.
Read In
(0) to (49)
Search for
Data Loc.
(1)
Data to
Distributor
(0) or (1)
Wait for
Even
(2)
Distributor
to Accum.
(0) or (2)
Compliment
Neg. Sum
If Req.
(1)
Remove
Interlock A
(1)
Restart
Signal
(1)
IA to AR
(1)
Enable PR
Read In
(0) to (49)
Search for
next Instr.
Multiply
Divide
14, 19, 64
(1)
Enable Distr.
Read In
(0) to (49)
Search for
Data Loc.
(1)
Data to
Distributor
(0) or (1)
Wait for
Even
(*)
Multiply or
Div. Loop
(1)
Remove
Interlock A
(1)
Restart
Signal
(1)
IA to AR
(1)
Enable PR
Read In
(0) to (49)
Search for
next Instr.
* Multiply Loop Requires (20) to (200)
Word Times.
Divide Loop Requires (60) to (240)
Word Times.
Shift Left
Shift Right
30, 35
(1)
Enable
Shift Contr.
(0) or (1)
Wait for
Even
(2)
Per Shift
(Shift Loop)
(1)
Remove
Interlock A
(0) or (1)
Restart Sig.
(unless
Shift 0)
(1)
IA to AR
(1)
Enable PR
Read In
(0) to (49)
Search for
next Instr.
Shift and
Count
36
(1)
Enable
Shift Contr.
(0) or (1)
Wait for
Even
(2)
Per Shift
(Shift Loop)
(1)
Remove
Interlock A
(0) or (1)
Restart Sign.
(unless $X_{20} \neq 0$)
(1)
IA to AR
(1)
Enable PR
Read In
(0) to (49)
Search for
next Instr.
Shift and
Round
31
(1)
Enable
Shift Contr.
(0) or (1)
Wait for
Even
(2)
Per Shift
(Shift Loop)
(2)
Round
(1)
Remove
Interlock A
(1)
Restart
Signal
(1)
IA to AR
(1)
Enable PR
Read In
(0) to (49)
Search for
next Instr.
continued

Figure 46A.

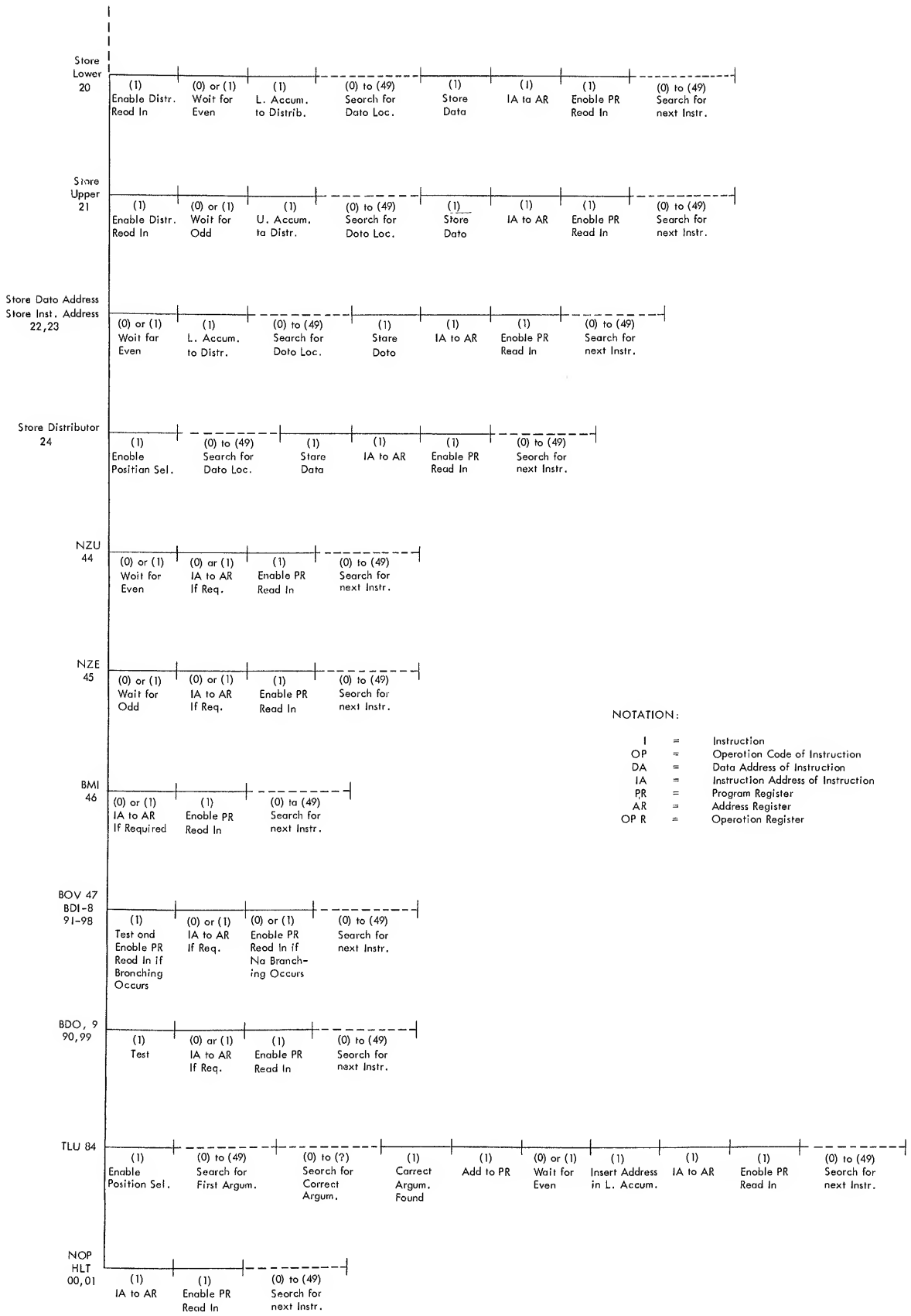

Store Lower 20
(1) Enable Distr. Read In
(0) or (1) Wait for Even
(1) L. Accum. to Distrib.
(0) to (49) Search for Data Loc.
(1) Store Data
(1) IA to AR
(1) Enable PR Read In
(0) to (49) Search for next Instr.
Store Upper 21
(1) Enable Distr. Read In
(0) or (1) Wait for Odd
(1) U. Accum. to Distr.
(0) to (49) Search for Data Loc.
(1) Store Data
(1) IA to AR
(1) Enable PR Read In
(0) to (49) Search for next Instr.
Store Data Address Store Inst. Address 22,23
(0) or (1) Wait for Even
(1) L. Accum. to Distr.
(0) to (49) Search for Data Loc.
(1) Store Data
(1) IA to AR
(1) Enable PR Read In
(0) to (49) Search for next Instr.
Store Distributor 24
(1) Enable Positian Sel.
(0) to (49) Search for Data Loc.
(1) Store Data
(1) IA to AR
(1) Enable PR Read In
(0) to (49) Search for next Instr.
NZU 44
(0) or (1) Wait for Even
(0) or (1) IA to AR If Req.
(1) Enable PR Read In
(0) to (49) Search for next Instr.
NZE 45
(0) or (1) Wait for Odd
(0) or (1) IA to AR If Req.
(1) Enable PR Read In
(0) to (49) Search for next Instr.
BMI 46
(0) or (1) IA to AR If Required
(1) Enable PR Read In
(0) to (49) Search for next Instr.
BOV 47 BDI-8 91-98
(1) Test and Enable PR Read In if Branching Occurs
(0) or (1) IA to AR If Req.
(0) or (1) Enable PR Read In if No Branching Occurs
(0) to (49) Search for next Instr.
BDO, 9 90,99
(1) Test
(0) or (1) IA to AR If Req.
(1) Enable PR Read In
(0) to (49) Search for next Instr.
TLU 84
(1) Enable Position Sel.
(0) to (49) Search for First Argum.
(0) to (?) Search for Correct Argum.
(1) Correct Argum. Found
(1) Add to PR
(0) or (1) Wait for Even
(1) Insert Address in L. Accum.
(1) IA to AR
(1) Enable PR Read In
(0) to (49) Search for next Instr.
NOP HLT 00,01
(1) IA to AR
(1) Enable PR Read In
(0) to (49) Search for next Instr.
NOTATION:
I = Instruction
OP = Operation Code of Instruction
DA = Data Address of Instruction
IA = Instruction Address of Instruction
PR = Program Register
AR = Address Register
OP R = Operation Register

Figure 46B.

Program Loading

After program writing is completed, each instruction is punched into a card. These cards (load cards) are then used to enter the program into the 650. Load cards can include not only the instructions, but also the constants, tables, etc., used to process the data.

Pressing the computer-reset key conditions the machine to get the next instruction from the storage-entry switches (8000). If the LOAD hubs on the input unit control-panel are activated by a 12-impulse, the next instruction location, after a read instruction, is specified by the D-address of the read command. Both of the above facts enter into the understanding of loading routines.

One Instruction Per Card

Each load card has punched in it a two-step program and the data (instruction to be loaded) that the program is to operate on. In addition, each card is designated a load card by control-panel wiring. One instruction is used to move the data (instruction being loaded) from the read-in area to the distributor. The other instruction moves the data from the distributor to another storage location (assigned by the programmer when the program is written). One additional instruction is necessary to complete the program. It is a RD command that is used to enter the other two instructions and the data into the machine. This is usually set-up in the storage-entry switches (8000).

To illustrate the action of this loading routine two of the instructions from the program in Figure 45 are loaded. Figure 47 shows the loading program and two standard load cards (Form 853862X), punched to load the RAL (0040) and LDD (0048) instructions.

Card columns 1-10 contain the instruction that moves the data (columns 31-40) from the read-in area to the distributor. Card columns 11-20 are used for indicative information. Card columns 21-30 contain the instruction that moves the data (columns 31-40) from the distributor to the storage location assigned by the programmer. Card columns 31-40 contain the data (instruction being loaded) that the other two instructions operate upon.

The instruction being loaded must have a sign punch over the units position. Card columns 41-45 contain the operation abbreviation. The remainder of the card columns can be used for indicative information as desired. This card has been designed for punching directly from the 650 Planning Chart Form X24-6151.

The read-in area (1951-1960) is chosen because none of the instructions being loaded are assigned to these locations. This in turn determines the address to be punched in columns 3-6 (1954) and 7-10 (1953) of each load card. Further, the choice of read-in area also determines the setting of the storage-entry switches (70-1951-xxxx). If a different read-in area is chosen, the punching in columns 3-10 of the load card would be changed to the addresses of the 4th and 3rd words of the selected read-in area. Also, the D-address portion of the RD instruction in 8000 would be changed to the 1st word of the read-in area.

Assume that cards A and B in Figure 47 are the first of a group of load cards to be entered into the machine. The load cards are placed into the read hopper of the input-output machine; the computer reset key is pressed and then the start key on the input-output device is pressed. As a result:

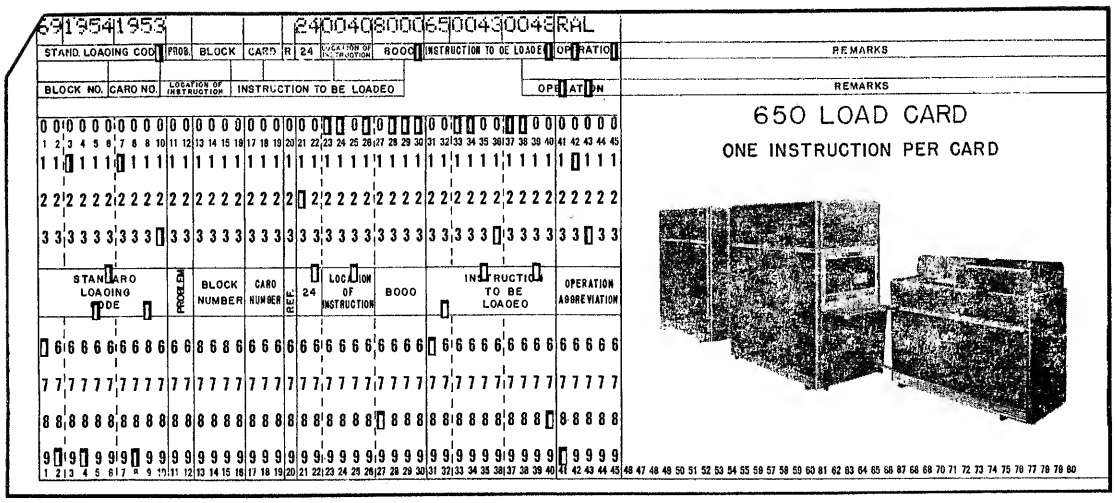
1. Three cards are run-in.
2. As the first two cards (A-B) pass first reading, the 12-punch in column 45 indicates, through control panel wiring, that these are *load* cards.
3. The first card (A) is automatically entered into the input synchronizer; columns 1-10 into word 1; columns 11-20 into word 2; columns 21-30 into word 3; etc.

The program-start key is then pressed to begin loading. Because the computer-reset key was pressed, the location of the first instruction is 8000. Instruction execution proceeds as follows:

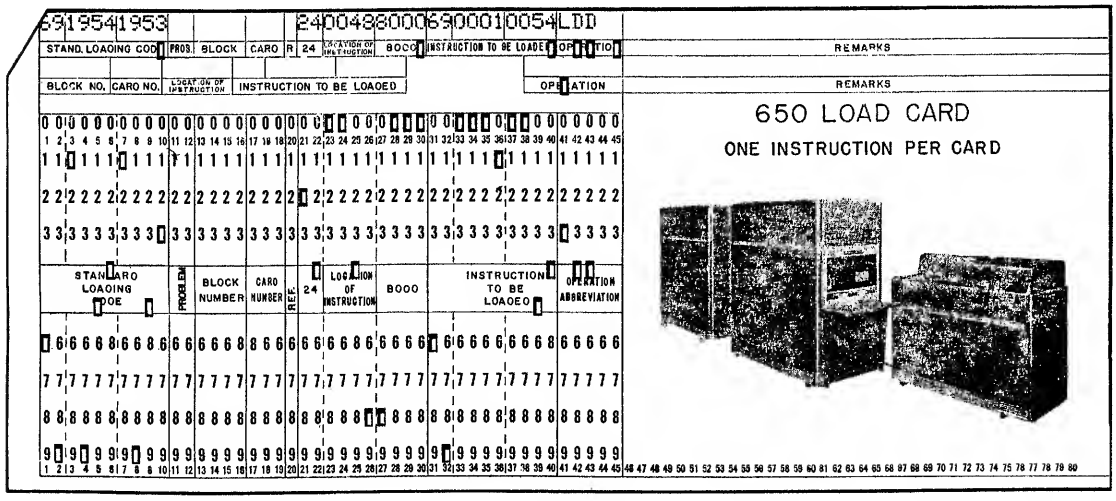
1. The first instruction (70-1951-xxxx) transfers the information (card A) from the input synchronizer to general storage and also initiates a mechanical cycle to refill the input synchronizer.
2. Because the LOAD hub was impulsed, the next instruction is indicated by the D-address (1951) of the RD command. This instruction (69-1954-1953) was entered into the machine when the information from card A was transferred from the input synchronizer to general storage.
3. The 69 (LDD) command moves the data (65-0043-0048) from storage word 1954 to the distributor. The data word must have a sign punched over its units position to prevent a validity-error.
4. The next instruction (24-0040-8000), located in word 1953, moves the data (65-0043-0048) from the distributor to storage location 0040. This is the location assigned by the programmer.
5. The program then returns to the storage-entry switches (8000), and the process is repeated for Card B. This will load the next instruction (69-0001-0054) into location 0048. The reading and loading will continue until all instructions are loaded.

The cards can be arranged in any desired order, depending on the extent of the indicative information.

Location of Instruction	Instruction			Operation Abbrev.	
	OP	Data	Instruction		
8000	70	1951	XXXX	RD1	A
1951	69	1954	1953	LDD	A
1953	24	0040	8000	STD	A
8000	70	1951	XXXX	RD1	B
1951	69	1954	1953	LDD	B
1953	24	0048	8000	STD	B



CARD A



CARD B

Figure 47. Loading Program and Cards

The cards can be sorted, collated, or listed in any desired manner for analyzing the program.

Because of its simplicity and flexibility, this method is suggested as the basic means for loading programs that are to be tested. Once a program has been tested and proved correct, it is a simple matter to convert the program to a multiple-instruction-per-card routine to save time in reloading. The original single-item

cards also provide the easiest method of reproducing the program by simple listings.

More Than One Instruction Per Card

It may be desirable, after a program has been proven to reduce the number of cards necessary to load the program. This can be accomplished by placing several instructions in each load card. Each card can contain up to eight instructions. Because it is usually desirable to have some means of identification punched into all program load cards, something less than eight instructions per card is recommended. Note: any loading method using more than two instructions per card requires storing a loading routine on the drum.

Four Instructions per Card

This routine enters four instructions from a card. In addition, the card contains the addresses of the locations into which the instructions are to be stored. The loading routine consists of nine instructions, five of which are stored on the drum, and four that are entered from each card as it is read. The program consists of a read instruction, followed by four sets of load and store distributor instructions. The four store-distributor instructions are punched into the cards, with the data address of each as the location into which the adjacent word is to be loaded.

The five instructions to be stored on the drum can be loaded from single-item load cards. A sixth card could then be used to effect the switch into the four-per-card routine just loaded.

After all the load cards have been processed, it is necessary to transfer to the main program just loaded. This can be accomplished in one of several ways:

1. If the first card following the program deck is not a load card, the switch to the main program can be accomplished by using the I-address of the read instruction.
2. Another method of switching to the main program is to change the I-address of the last store-distributor instruction in the last program load card. The I-address used would simply be the location of the first instruction in the main program. By proper choice of the I-address of the last store-distributor instruction in any load card, cards containing one, two, or three instructions can also be processed by the four-per-card loading routine.

Figure 48 is an example of a routine for loading four instructions per card, using locations 1951-1960 as the read-in area.

Location of Instruction	Instruction			Operation Abbrev.	
	OP	Data	Instruction		
1999	70	1995	0000	RD1	*
1995	69	1952	1951	LDD	*
1951	24	XXXX	1996	STD	**
1996	69	1954	1953	LDD	*
1953	24	XXXX	1997	STD	**
1997	69	1956	1955	LDD	*
1955	24	XXXX	1998	STD	**
1998	69	1958	1957	LDD	*
1957	24	XXXX	1999	STD	**

*These instructions kept on the drum.
**These instructions are read into the 650 on each load card. The data addresses specify the location into which each of the four pieces of information is to be stored.

Figure 48. Four Instructions per Card

Seven Instructions per Card

This routine is used to load as many as seven instructions from a single load card. The eighth word is used as a control word and is punched with the address into which the first instruction is to be entered and the number of instructions to be entered from that particular card. Because only one address is in each card, the instructions from each card are loaded into consecutive drum locations.

Figure 49 shows a routine to load as many as seven instructions per card, using the control-word layout just mentioned.

Location of Instruction	Instruction			Operation Abbrev.	
	OP	Data	Instruction		
1988	70	1994	M	RD1	(Transfer control to location M if no load card).
1994	65	1951	1979	RAL	
1979	69	1982	1986	LDD	
1986	22	1982	1989	SDA	
1989	35	0004	1981	SLT	Prepare to store starting with location F.
1981	15	8001	1990	ALO	
1990	22	1978	1984	SDA	
1984	65	1991	1992	RAL	
1992	10	1982	8002	AUP	
8002	69	(1952)	8003	LDD	Store word I in location F+I−1.
8003	24	(F)	1977	STD	
1977	10	1980	1985	AUP	Modify instructions in preparation to store word I+1.
1985	15	8001	1993	ALO	
1993	11	1978	1983	SUP	Test to see if word N−1 has been stored. If so, read the next card.
1983	44	1987	1988	NZU	
1987	10	8001	8002	AUP	Prepare to store word I+1.
1982	24	0000	1977	STD	Constants
1991	69	1952	8003	LDD	
1980	00	0001	0000		
1978	24	F+N	1977	STD	Temporary storage
1951	00	F	000N		Control word punched in load card F is address where first word on card is to be entered. N is the number of words to be entered.

Figure 49. Seven Instructions per Card

Index

Absolute Arithmetic Codes 20
Add Magnitude to Lower Accumulator (17-AML) 20
Add to Lower Accumulator (15-ALO) 2
Add to Upper Accumulator (10-AUP) 3
Argument (TLU) 16
Arithmetic Operation Codes 1
Available Computing Time 22
Branching Operation Codes 12
Branching Function of Read Codes 16
Branch on Digit Eight in a Distributor Position (90-99-BD 0-9) 15
Branch on Minus Accumulator (46-BMI) 15
Branch on Non-Zero Accumulator (45-NZE) 15
Branch on Non-Zero in Upper Accumulator (44-NZU) 13
Branch on Overflow (47-BOV) 15
Divide (14-DIV) 6
Divide and Reset Upper Accumulator (64-DVR) 6
Four Instructions per Card Program Loading 34
Function (TLU) 16
Independent Accumulator Operations 4
Input-Output Unit Interlocks 22
Interlock Time — 533 22
Interlock Time — 537 22
Interlock Time — 407 22
Load Distributor (69-LDD) 16
Miscellaneous Operation Codes 16
Multiply (19-MPY) 5
No Operation (00-NOP) 16
One Instruction per Card Program Loading 32
Operation Codes 1
Optimum Programming 21
Optimum Programming Accumulator Considerations 22
Optimum Program Chart 23, 24
Optimum Programming Techniques 21
Optimizing Arithmetic Codes 24
Optimizing Branching Codes 25
Optimizing Shifting Codes 25
Optimizing 800X Addresses 26
Optimizing 8000 26
Optimizing 8001 26
Optimizing 8002 26
Optimizing 8003 27
Program Loading 32
Programming from Fixed Locations 27
Reset and Add to Lower Accumulator (65-RAL) 2
Reset and Add to Upper Accumulator (60-RAU) 3
Reset and Add Magnitude to Lower Accumulator (67-RAM) 20
Reset and Subtract from Lower Accumulator (66-RSL) 4
Reset and Subtract Magnitude from Lower (68-RSM) 20
Reset and Subtract from Upper Accumulator (61-RSU) 4
Sequence Chart 28
Seven Instructions per Card Program Loading 34
Sign Analysis 1
Shift Operation Codes 10
Shift Left (35-SLT) 11
Shift Left and Count (36-SCT) 11
Shift Right (30-SRT) 11
Shift Right and Round (31-SRD) 11
Stop-Halt (01-HLT) 16
Store Operation Codes 8
Store Data Address of Lower Accumulator (22-SDA) 9
Store Distributor (24-STD) 10
Store Instruction Address of Lower Accumulator (23-SIA) .. 10
Store Lower Accumulator (20-STL) 8
Store Upper Accumulator (21-STU) 9
Subtract from Lower Accumulator (16-SLO) 4
Subtract Magnitude from Lower Accumulator (18-SML) 20
Subtract from Upper Accumulator (11-SUP) 4
Table Lookup 16
Table (TLU) 16
Table Lookup (84-TLU) 18
TLU End-of-Table — End-of-Search 18
TLU General Operation 18
TLU Other Considerations 19
TLU Program Example 19

IBM

International Business Machines Corporation
590 Madison Avenue, New York 22, N. Y.

Printed in U.S.A. G24-5002-0 9/58